Marquer comme terminé

Préliminaires

1. télécharger les données : https://ecampus.unicaen.fr/mod/resource/view.php?id=817063
2. insérer chaque fichier BSON à l'aide de la commande :

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

Répondre à une requête utilisateur

3. Écrire une requête *pure Mongo* qui affiche les identifiants des pages contenant les mots "poste machine learning" en n'interrogeant que la collection des pages.

Expérimenter un script JS dans Mongo

Expérimenter un script javascript comme suit (adapter le script sur votre exemple pour itérer avec un curseur, noter que la base de données n'est pas précisée, elle le sera à l'exécution) :

```
function printDoc(doc) {
    doc._id = doc._id.valueOf();
    print(doc._id)
}

cursor = db.terms.find({...<préciser ici votre requête>...});
// ou bien
cursor = db.pages.find({...<préciser ici votre requête>...});

if (cursor.hasNext()) {
    printDoc(cursor.next());
    while (cursor.hasNext()) {
        printDoc(cursor.next());
    }
}
```

Un script JS s'exécute en le redirigeant vers les entrées de l'interpréteur Mongo

```
mongosh --quiet <database> -f script.js
```

Pour afficher les statistiques d'exécution, ajouter la méthode suivante à la requête find :

```
    .explain("executionStats")
```

4. Obtenir la même réponse qu'à la question 3 (les identifiants des pages contenant les termes de la requête) en valorisant l'index : la réponse à la requête est l'intersection des index des mots de la requête. On pourra s'aider du canevas fourni : https://ecampus.unicaen.fr/mod/resource/view.php?id=817067.

Il manque :

- le calcul de la norme d'un vecteur
- le calcul du cosinus

5. Comparer les temps d'exécution

- en ajoutant `.explain("executionStats")` à la requête sur les pages et examinant le champ "executionTimeMillis" : 182ms
- en chronométrant la requête qui procède à l'intersection des index :

?

```
$ time -p mongo --quiet engine engine.js
35298
{query:poste machine learning}
96,139,203,759,1344,1730,1739,1752,1758,1777,1780,1781,1783,1791,1824,1825,1834,1865,2055,2305,2355,2495,2969,2979,2990,3114,3
126,3128,3166,3270,3272,3570,3636,3638
real 0,06
user 0,04
sys 0,01
```

Déterminer la page la plus pertinente pour la requête

- on associe à la requête un vecteur constitué des DF de chaque mot
- à chaque document contenant les mots de la requête, on associe un vecteur des TF-iDF des mots de la requête
- la page la plus pertinente est celle qui obtient le meilleur `cosinus` entre son vecteur et le vecteur de la requête.

Résultat à obtenir :

```
tp@tp-bdd:~/mongo$ mongo --quiet engine engine.js
{query:poste machine learning}
the result is :
(page) => (cosinus)
96 => 0.9568857073533784
139 => 0.9568856146126343
203 => 0.9592576637590551
759 => 0.9592576637590551
1344 => 0.7813272483113842
1730 => 0.8376853265066341
1739 => 0.8376853265066341
1752 => 0.9269101891391914
1758 => 0.7469081190094149
1777 => 0.7505853641589632
1780 => 0.8483139640600882
1781 => 0.9928616401515897
1783 => 0.9928616401515897
1791 => 0.9961457046945343
1824 => 0.956885568241751
1825 => 0.9568857073533784
1834 => 0.9961457046945343
1865 => 0.9568857073533784
2055 => 0.9568857073533784
2305 => 0.9568857073533784
2355 => 0.9568857073533784
2495 => 0.9592576637590551
2969 => 0.9568857073533784
2979 => 0.9568857073533784
2990 => 0.9568857073533784
3114 => 0.8376853265066341
3126 => 0.7505853641589632
3128 => 0.9928616401515897
3166 => 0.9692617760189819
3270 => 0.956885568241751
3272 => 0.9568857073533784
3570 => 0.9961457046945343
3636 => 0.9928616401515897
3638 => 0.9961457046945343
```

Indications

Exécution d'un script JS par **mongo**

Le script suivant effectue une requête qui retourne un curseur. On peut ensuite itérer sur le curseur.

```
cursor = db.terms.find({$or: [{_id:"poste"},{_id:"machine"},{_id:"learning"}]})

if (cursor.hasNext()) {
    print(cursor.next()._id);
    while (cursor.hasNext()) {
        print(cursor.next()._id);
    }
}
```

On exécute cette requête par\ :

```
    mongosh --quiet localhost/engine requete.js
```

?

Génération d'une requête dynamique

On souhaite répondre à la requête `poste machine learning` de l'utilisateur\ :

```
query = "poste machine learning"
or = query.split(" ").map(word => {return {_id: word}})
print(JSON.stringify(or))

cursor = db.terms.find({$or: or})
```

Intersection des tableaux de page

Fonction de calcul d'intersection

```

/ returns the intersection between two arrays
function intersection(tab1, tab2) {
// assuming tab1 and tab2 are sorted
    var ret = [];

    var i = 0;
    var j = 0;

    while (i < tab1.length && j < tab2.length) {
        if (tab1[i] == tab2[j]) {
            ret.push(tab1[i]);
            i++;
            j++;
        } else if (tab1[i] < tab2[j])
            i++;
        else
            j++;
    }
    return ret;
}
```

Intersection par accumulation

```
if (cursor.hasNext()) {
    resultat = cursor.next().index;
    while (cursor.hasNext()) {
        resultat = intersection(resultat, cursor.next().index);
    }
}

print(resultat.length)
```

On vérifie qu'on trouve 34 pages, ce qui correspond au résultat des requêtes

```
db.pages.find({$and:[{"words.word":"poste"},{"words.word":"machine"},{"words.word":"learning"}]}).count()
db.pages.find({"words.word":{$all:["machine", "learning", "poste"]}}).count()
```

Si on ajoute `.explain("executionStats")`, on constate que le `$all` prend 118ms et le `$and` prend 113ms.

En revanche :

```
time -p mongo --quiet localhost/engine engine.js
[{"_id":"poste"},{"_id":"machine"},{"_id":"learning"}]
34
real 0,04
user 0,03
sys 0,01
```

Fin du travail

Il reste maintenant à calculer le cosinus de l'angle entre chaque page qui matche et la requête. Pour cela, on doit avoir

1. le vecteur de la requête : c'est la liste des nombres d'occurrences de chaque mot dans le corpus donc c'est la liste des longueurs d'index de chaque (voir méthode `computeQueryVector`)
2. le vecteur des TF-iDF de chaque page : il faut aller, dans chaque page qui matche, rechercher le TF-iDF de chaque mot de la requête (voir méthode `computeVector`)

Un cannevas assez fourni est disponible ici.

Conclusion

?

- `mongo` dispose d'un langage puissant de requêtage. La méthode `find` permet des sélections souples mais pas forcément efficace.

- `mongo` est capable d'interpréter du Javascript, ce qui permet d'écrire des scripts de requêtage très évolués.
- en l'absence de jointure, il est nécessaire d'effectuer des pré-calculs annexes (ici, les index) pour améliorer la performance des opérations.

Modifié le: jeudi 24 octobre 2024, 11:49

◄ TP 6.1

Choisir un élément

Aller à...

Cannevas pour TP 6.2 ►

?