Alexis MAREMBERT
Théophile WALTER

Compte Rendu Projet - TweetStats

Langages dynamiques - Master 1 ISD

Sommaire

Introduction

A. Rappel du projet

Nous avons réalisé une interface graphique de reporting permettant de faire un rendu graphique synthétique de statistiques sur un ensemble de tweets.
Ces messages proviennent de la plateforme Twitter. L'application contient deux parties, une client et l'autre serveur. Ces deux parties interagissant via le protocole HTTP.

Le serveur réalisé entièrement en python est capable de servir des fichiers, gérer des sessions HTTP, reconnaître certaines URL spéciales et déclencher l'exécution de code Python arbitraire lorsqu'on navigue vers ses URLs, mettre à disposition du code Python exécuté un objet représentant la session HTTP ainsi que les paramètres de la requête HTTP.
Le serveur utilisant du spark est également capable d'importer un fichier JSON contenant les tweets dans sa base, d'effectuer des requêtes sur sa base et de la supprimer.

Le client est entièrement implémenté en pur CSS/HTML/Javascript et fait des requêtes asynchrones au serveur. Il s'agit de l'interface graphique proposant une barre de recherche pour la saisie de mots-clés. L'interface permet de visualiser à l'aide de graphiques le nombre de tweets trouvés, d'une carte du monde la répartition par pays, et un nuage de mots des hashtags et mots-clés associés à ces tweets.

B. Répartition du travail

L'utilisation de GitHub pour travailler en binôme sur l'application nommée "TweetStats" nous a paru comme une évidence. Le langage choisi pour développer l'application est l'anglais. Nous avons décortiqué le sujet et sommes convenus de nous répartir les tâches de façon symbolique.

Théophile a commencé par dessiner l'architecture de l'application afin de mieux visualiser les tâches à effectuer. Il s'est ensuite chargé du serveur web en Python, des asynchrones en Javascript ainsi que l'affichage en Javascript. Alexis tant qu'à lui s'est chargé du requêtage en Python et de l'affichage en Javascript.
La répartition du code ne nous a pas empêché d'intervenir dans les parties de l'un et l'autre pour assurer et améliorer la parfaite communication de nos parties.

I. Architecture de l'application

A. Choix d'architecture

L'écriture du fichier "README.md" pour visualiser et indiquer l'avancement des travaux a été la première étape pour connaître précisément les besoins et les objectifs à atteindre. Cette liste nous est avérée très utile au bon déroulement du projet.

Nous avons choisi de travailler avec Python 2.7.12, Spark 2.2.1, Hadoop 2.7 et un navigateur web tel que Firefox. Pour utiliser le serveur nous avons utilisé le port 2319 de la machine locale.

L'architecture de l'application "TweetStats" se compose en trois parties. Premièrement, la donnée en JSON est disponible via une URL et à télécharger dans ce dossier. Deuxièmement, le back-end contenant les fichiers du serveur et de la recherche en Python. Troisièmement, le front-end qui se compose des fichiers HTML et CSS de la page d'accueil de l'application, mais également dans d'un sous dossier les images et un autre les classes Javascript pour ces fichiers.

B. Back-end

L'architecture du back-end est composée de deux fichiers, le serveur et la recherche. Le serveur "back/server.py" est une implémentation de la classe python SimpleHTTPServer.

Ce serveur renvoie par défaut les fichiers présents dans le répertoire "front/" comme n'importe quel serveur HTTP. Mais il analyse également les requêtes du client de façon à traiter à part les requêtes dont le chemin d'accès commence par "/api/". Ces requêtes ne sont donc pas de réels fichiers simplement renvoyées. Une requête commençant par "/api/search/" indique que le client souhaite effectuer une nouvelle recherche, cela déclenche l'exécution du script python/spark "back/search.py" (dont le fonctionnement est résumé ci-dessous), cette requête doit retourner un identifiant unique de recherche au client pour que celui-ci puisse demander le résultat de la recherche ultérieurement. Une requête commençant par "/api/get-response/" indique que le client demande au serveur si le résultat de sa recherche est prêt ou non. Le serveur répond le cas échéant soit un message d'attente, soit la réponse.

Le recherche est gérée par le fichier "back/search.py" qui est un script python pour l'environnement Spark. Celui-ci charge le JSON en mémoire et l'analyse pour pouvoir l'utiliser. Il est stocké dans une RDD où chaque ligne contient un dictionnaire représentant l'objet JSON correspondant à un tweet. De cette manière, grâce aux opérations fournies par Spark sur les RDD (map, filter, reduceByKey, etc.), il est très simple d'effectuer des recherches et des statistiques sur la base de données.

C. Front-end

L'architecture se compose tout d'abord d'un fichier "graph-test.html" qui nous a servi à mettre nos idées en communs et à tester les différentes interfaces avant de les intégrer à la véritable page. Le fichier "index.html" est la page officielle de notre application. On y ajoute au fur et à mesure les codes qui ont passé avec succès les tests dans le fichier précédent. Nous avons choisi de répertorier dans un seul fichier "style.css" les règles de styles. Dans le dossier "images" nous mettons uniquement nos images sous format SVG pour un meilleur rendu visuel. Finalement, nous mettons les classes Javascript utilisées par les fichiers HTML dans le dossier "classes", pour une meilleure cohérence dans l'arborescence de nos fichiers.

Tous les éléments graphiques utilisés implémentent une des deux classes mères, à savoir "Graph" ou "HtmlDisplay" suivant s'il s'agit respectivement d'un affichage dans un canvas ou d'un véritable code HTML. Cela permet de mettre en commun dans ces classes mères les codes redondants, comme la création du canvas, ou la sauvegarde des données si nous avons besoin de redessiner ces rendus.

La classe "ResizableGraph" permet de créer simplement un affichage (implémentant la classe "Graph") redimensionnable à la souris. Nous verrons dans la partie suivante un exemple concret.

II. Exemples judicieux de code complexe

A. Back-end

Nous allons maintenant montrer quelques exemples de code complexe que nous jugeons judicieux de présenter côté back-end. Tout d'abord voici un code permettant de compter l'occurrence d'un mot dans le résultat d'une recherche. "rdd_one_word" est une RDD contenant les tweets répondant à une recherche demandée.

```
# Compte les mots les plus utilisés
word_count = rdd_one_word.flatMap(lambda line: re.split('[  ,.;:!?]',
line['text'])) \
            .map(lambda word: (word, 1)) \
            .reduceByKey(lambda a, b: a + b) \
            .map(lambda e: (e[1], e[0])) \
            .sortByKey(ascending=False)
```

Dans ce code, on commence par récupérer tous les modes dans tous les textes des tweets sélectionnés, ensuite on associe à chaque mot la valeur 1, ce qui va nous permettre de "regrouper" tous les mots identiques en faisant à chaque fois la somme des deux valeurs. On se retrouve donc avec comme résultat le compte du nombre d'occurrences de chaque mot. Ensuite, on trie la RDD par ordre décroissant de façon à pourvoir par la suite sélectionné les N premier mots les plus utilisés.

B. Front-end

Après avoir vu un exemple de code back-end, nous allons maintenant voir du code front-end complexe. Voici par exemple deux codes équivalents montrant l'utilité de ce système d'héritage qui nous permet de diminuer le nombre de lignes et d'améliorer la lisibilité du code.

Exemple sans l'utilisation de l'héritage (dessin d'un histogramme redimensionnable) :

```
<!-- Canvas redimensionnable →
    <div class="resizeBox" style="width:500px;height:250px;"
onmouseup="javascript:histo.resize(this.clientWidth-15, this.clientHeight-15);">
     <canvas id="graph-1"></canvas>
  </div>
  <script type="text/javascript">
    // Crée un histogramme simple
    var g1 = document.getElementById("graph-1");
    var histo = new Histogram(g1);
    histo.setBackgroundColor("#F0F0F0");
    histo.setForegroundColor("#AA0000");
    // Quelques données
    var d1 = [
      ["Data 1", 40],
      ["Data 2", -12, 10, "#00AA00"],
      ["%phantom%", 100], // Espace et hauteur max
      ["Data 3", 89, "yellow"]
    ];
    // On dessine
    histo.draw(d1);
  </script>
```

Voici le même code mais cette fois-ci en utilisant le système d'héritage :

```
  <script type="text/javascript">
    var graph = new ResizableGraph(500, 250, document.body, Histogram);
    graph.setBackgroundColor("#F0F0F0");
    graph.setForegroundColor("#AA0000");
    graph.draw(d1);
  </script>
```

III. Structures de données utilisées

A. Back-end

Le fichier “search.py” utilise des RDD qui contientent des dictionnaires représentant le JSON des tweets. Nous avons jugé judicieux de procéder ainsi car l’extraction de l’information est beaucoup plus simple une fois la donnée sous ce format, en effet cela nous permet d’accéder aux attributs directement via leur nom, ce qui rend le code beaucoup plus simple à écrire et à comprendre.

B. Front-end

Contrairement au back-end nous utilisons uniquement les données sous format JSON ainsi que des tableaux javascript classiques. En effet l’utilisation de JSON en JavaScript permet une très grande flexibilité ainsi qu’une intégration parfaite dans le langage.

IV. Bugs rencontrés

A. Back-end

Nous allons maintenant voir les problèmes auxquels nous avons été confrontés et que nous jugeons les plus intéressants. Cependant, nous n’avons rencontré aucun problème notable côté back-end. Contrairement au côté front-end que nous allons aborder dans le point suivant.

B. Front-end

Côté front-end nous avons rencontré des difficultées lors de la création de la carte du monde. Tout d’abord nous voulions une carte avec une projection de Mercator. Nous étions parvenus à placer les points plus ou moins de façon précise. Cependant nous avions toujours le problème aux extrémitées nord-ouest, nord-est, sud-ouest et sud-est. Les points étaient en effet mal placés. Après maints efforts nous avons décidé de changer de projection et de choisir celle de Lambert qui est une projection équivalente cylindrique. Cette projection nous a permis de coder beaucoup plus rapidement la fonction qui permet de mettre sur la carte les points de coordonnées GPS.

Nous avons également eu des difficultés avec les créations dynamiques d'éléments, en effet lorsque nous utilisions “element.innerHTML” pour modifier/créer des éléments, il arrivait que certains attributs d’autres éléments disparaissent (comme des onclick). La solution a donc été de séparer chaque vue dans des balises bien différentes de façon à cloisonner les modifications et ainsi éviter qu’elles aient des répercussions non souhaitées.

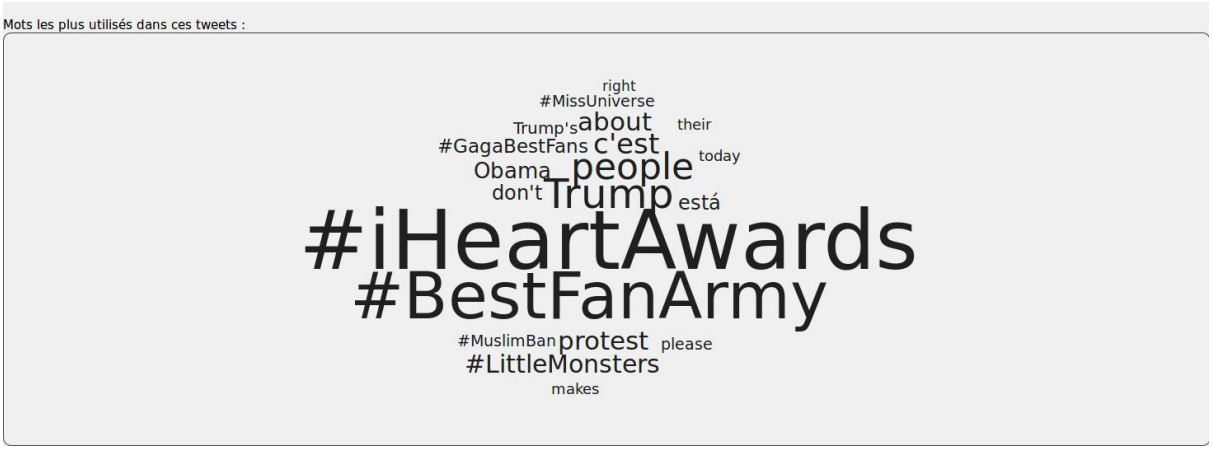
Conclusion

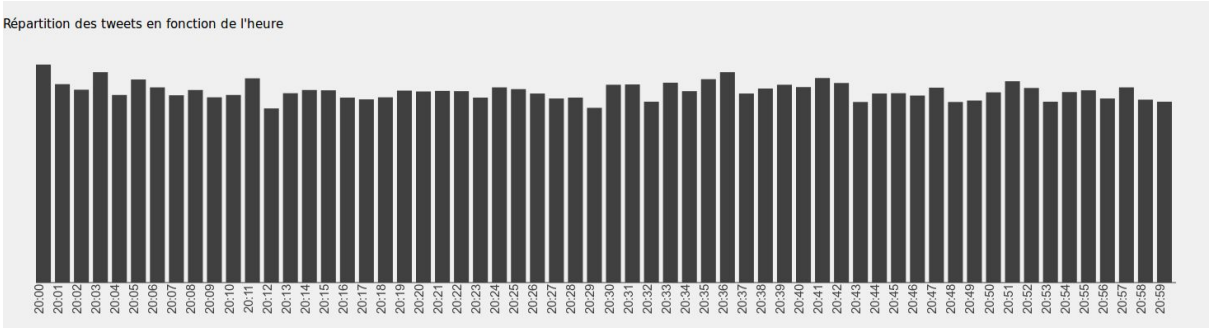
Pour conclure ce compte rendu de projet, nous pouvons dire que nous avons abordé tous les points demandés dans le sujet. De plus, nous avons également ajouté des éléments supplémentaires qui nous paraissaient intéressant. Le projet est complet et disponible avec l'archive "TweetStats-master.zip".

Nous retenons surtout l'expérience acquise en travaillant en binôme, ainsi que des moments de discussion sur comment aborder un problème et quel est la meilleure solution de pour nous en fonction de notre besoin spécifique.

Annexes


Localisation des tweets (116 tweets localisés)


Mots les plus utilisés dans ces tweets :
right
#MissUniverse
Trump's
about
their
#GagaBestFans
c'est
today
Obama
people
don't
Trump
está
#iHeartAwards
#BestFanArmy
#MuslimBan
protest
please
#LittleMonsters
makes


Répartition des tweets en fonction de l'heure

Résultat brut


```
{
   "status" : "1",
   "id" : "92cd1551c7184d7fd813d970ffc76cfc",
   "ready" : "1",
   "result" : {
      "countOneOf" : "39341",
      "countAll" : "0",
      "coords" : [ ... ],
      "mostUsedWords" : [
         [ ... ],
         [ ... ],
         [ ... ],
         [ ... ],
         [
            "please",
            18.930547019053474,
            308
         ],
         [ ... ],
         [ ... ],
         [ ... ],
         [
            "#GagaBestFans",
            23.232944068838354,
            378
         ],
         [ ... ],
         [ ... ],
         [ ... ],
         [
            "#LittleMonsters",
            29.502151198524892,
            480
         ],
         [ ... ],
         [ ... ],
         [ ... ],
         [ ... ],
         [ ... ],
         [ ... ],
         [ ... ]
      ],
      "tweetsByTime" : [ ... ]
   }
}
```


Termes de la recherche

ceci est ma recherche

Tweets contenant tous ces mots : 0

Tweets contenant au moins un de ces mots : 39341