01/09/2022

Tâches pratiques

# Module 1 ( 13.02.2023 - 13.03.2023 ) \_ \_ \_ \_ \_ \_

Développement du gestionnaire d'analyseur

## Conception et tâche pratique du module

### Développement et test d'un parseur pour le traitement des pages html .

### Développement et test d'un parseur pour le traitement automatisé de formats de documents. pdf , . doc , . docx , djvu .

## Sujets des séminaires du module :

### Méthodes modernes de classement des résultats de recherche d'informations sur le Web.

### Extraction des journaux de requêtes pour améliorer les opérations des moteurs de recherche Web.

### Des modèles langage (Bag of words, Word2Vec, Word embedding et etc. ) et méthodes traitement informations textuelles ( WordNet , lemmatisation , stemming ).

### Encodage des données textuelles basé sur le modèle GPT .

## Date limite du module - 13 . 0 3.202 3

# Module 2 (14.03.2023-10.04.2023)

Développement d'un robot de recherche pour la collecte et le traitement de données issues de ressources Web 1.0 / Web 2.0

## Conception et tâche pratique du module

### Développement du modèle le plus simple d'un robot de recherche avec un algorithme classique de collecte et de traitement de données dans le réseau Web 1.0/Web 2.0.

### automatisée ̆ utilisant le modèle le plus simple d'un robot de recherche basé sur un algorithme d'exploration spécialisé utilisant l'exemple des sites Web de l'Université d'État de Saint-Pétersbourg et de l'Université d'État de Moscou - pour le Web 1.0 / Collecte de données automatisée basée sur l'API du réseau social VKontakte ou le Telegram messager sur les mentions dans les publications des utilisateurs de l'Université d'État de Saint-Pétersbourg et de l'Université d'État de Moscou - pour le Web 2.0 .

### Collecte de statistiques de pages traitées pour le Web 1.0 : nombre total de pages et de tous les liens, nombre de pages internes, nombre de pages cassées, nombre de sous-domaines internes, nombre total de liens vers des ressources externes, nombre de ressources externes uniques, nombre de liens vers des fichiers doc / docx / pdf . Statistiques pour le Web 2.0 : le nombre de publications sur la mention de l'université, le nombre d'utilisateurs publiant du contenu, le nombre de likes/vues/commentaires/reposts, le graphique du nombre de publications par jour pour la période collectée.

## Sujets des séminaires du module :

### Méthodes de mise à jour des données dans l'index à l'aide de robots de recherche. Stratégies de renouvellement uniforme et proportionnel.

### algorithme PageRank et ses modifications pour le calcul des poids des pages Web des sites.

### Mesures de centralité de graphe utilisées dans l'analyse des données des réseaux sociaux ̆.

### Apprentissage par renforcement ( Renforcement Apprentissage ) .

### Méthodes de réduction de la dimension des données.

## Date limite du module - 10.04.2023

# Module 3 (11.04.2023-15.05.2023)

Développement du modèle d'indice inversé le plus simple

## Conception et tâche pratique du module

### Implémentation et test d'une structure d'index basée sur un index inversé.

### Application de la méthode de compression de l'indice inversé à l'aide du codage Elias Delta et Gamma .

### Test du processus d'indexation sur les pages Web du site Web de l'Université d'État de Saint-Pétersbourg ou de l'Université d'État de Moscou collectées à la 2e étape (la vitesse du processus d'indexation par le nombre de documents texte est d'environ 40 000, pour vérifier l'efficacité de l'indexation à l'aide du l'algorithme de compression réduit la quantité d'informations stockées par rapport à la situation classique qui n'implique pas l'utilisation de l'algorithme de compression). Vérifiez la vitesse de recherche pour la requête "Recteur de l'Université d'État de Saint-Pétersbourg / Université d'État de Moscou".

## Sujets des séminaires du module :

### Indexation distribuée ( MapReduce ) et dynamique.

### Elias Delta et Gamma . Algorithmes de compression : un algorithme de codage Golomb paramétrique .

### suffixes de structure d'index , leurs principes de structuration de l'information et leurs caractéristiques architecturales.

### Tâche de déduplication des données

## Date limite du module – 15/05/2023

1,Développement et test d'un parseur pour le traitement des pages html

Le développement d'un parseur pour le traitement des pages HTML est une tâche complexe qui nécessite une compréhension approfondie des spécifications du langage HTML et des techniques de traitement de texte.

Le parseur doit être capable de lire le code HTML d'une page Web, de l'analyser et d'en extraire les informations pertinentes. Il doit également être capable de gérer les erreurs de syntaxe et de récupérer les données même lorsque la page HTML n'est pas correctement formatée.

Le processus de développement d'un parseur pour le traitement des pages HTML peut être divisé en plusieurs étapes :

1. Analyse des spécifications HTML : il est important de comprendre les spécifications HTML et de connaître les balises et les attributs couramment utilisés.
2. Définition des règles de grammaire : il est nécessaire de définir les règles de grammaire pour le parseur, qui décrivent les balises, les attributs et les éléments de la page HTML.
3. Programmation du parseur : une fois que les règles de grammaire ont été définies, le parseur peut être programmé en utilisant un langage de programmation tel que Python, Java ou C++. Le parseur doit être capable de lire le code HTML de la page Web, de l'analyser selon les règles de grammaire et d'en extraire les informations pertinentes.
4. Test et débogage : une fois le parseur développé, il doit être testé et débogué pour s'assurer qu'il fonctionne correctement et qu'il peut extraire les données souhaitées de toutes les pages HTML.

Lors du test du parseur, il est important de vérifier que les données sont correctement extraites et qu'elles ne sont pas corrompues. Il est également important de tester le parseur sur une variété de pages HTML pour s'assurer qu'il est capable de gérer différents formats et structures de page.

En conclusion, le développement d'un parseur pour le traitement des pages HTML est une tâche complexe qui nécessite une bonne compréhension des spécifications HTML et des techniques de traitement de texte. Le parseur doit être programmé pour lire le code HTML, l'analyser et en extraire les informations pertinentes, puis testé pour s'assurer qu'il fonctionne correctement sur une variété de pages HTML.

exemple

Voici un exemple simplifié de code Python pour un parseur HTML qui utilise la bibliothèque BeautifulSoup :

pythonCopy code

from bs4 import BeautifulSoup

import requests

# URL de la page HTML à traiter

url = 'https://www.example.com'

# Télécharger la page HTML

response = requests.get(url)

# Créer un objet BeautifulSoup pour l'analyse du code HTML

soup = BeautifulSoup(response.text, 'html.parser')

# Extraire le titre de la page HTML

title = soup.title.string

# Extraire tous les liens de la page HTML

links = soup.find\_all('a')

for link in links:

print(link.get('href'))

Dans cet exemple, nous utilisons la bibliothèque BeautifulSoup pour analyser le code HTML d'une page Web à l'aide de la méthode requests.get pour télécharger le code HTML de la page à partir de son URL. Nous créons ensuite un objet BeautifulSoup à partir du code HTML pour pouvoir extraire les informations pertinentes.

Dans ce cas, nous extrayons le titre de la page HTML en utilisant la méthode soup.title.string et tous les liens de la page en utilisant la méthode soup.find\_all('a'). Nous parcourons ensuite tous les liens de la page à l'aide d'une boucle for et utilisons la méthode link.get('href') pour extraire l'URL de chaque lien.

Bien sûr, un parseur HTML plus complexe peut inclure des fonctions supplémentaires pour extraire des informations spécifiques d'une page Web, mais cet exemple donne une idée de base de la façon dont un parseur peut être développé en Python en utilisant la bibliothèque BeautifulSoup.

Le développement d'un parseur pour le traitement automatisé de formats de documents tels que PDF, DOC, DOCX et DJVU peut être un processus complexe, mais il est possible d'utiliser des bibliothèques Python pour simplifier le processus. Voici un exemple de processus que vous pourriez suivre pour développer et tester un parseur pour ces formats de documents :

1. Installer les bibliothèques nécessaires

Pour le traitement de PDF, vous pouvez utiliser la bibliothèque PyPDF2, qui permet de lire, écrire et manipuler des fichiers PDF. Pour le traitement de fichiers DOC et DOCX, vous pouvez utiliser la bibliothèque python-docx, qui permet d'extraire le texte et les images des fichiers Word. Pour le traitement de fichiers DJVU, vous pouvez utiliser la bibliothèque PyDjVuLibre, qui permet de lire et d'écrire des fichiers DJVU.

1. Charger le fichier et extraire le texte

Pour chaque format de document, vous devrez charger le fichier dans le programme et extraire le texte pour le traitement. Pour les fichiers PDF, vous pouvez utiliser PyPDF2 pour extraire le texte. Pour les fichiers Word, vous pouvez utiliser python-docx pour extraire le texte et les images. Pour les fichiers DJVU, vous pouvez utiliser PyDjVuLibre pour extraire le texte.

1. Traiter le texte extrait

Une fois que vous avez extrait le texte du fichier, vous pouvez le traiter pour extraire les informations pertinentes. Vous pouvez utiliser des techniques de traitement de texte telles que la tokenisation, la lemmatisation et la reconnaissance d'entités nommées pour extraire des informations spécifiques du texte.

1. Tester le parseur

Pour tester le parseur, vous pouvez utiliser des fichiers d'exemple pour chaque format de document et vérifier si les informations sont correctement extraites. Vous pouvez également vérifier si le parseur fonctionne correctement avec des fichiers de différents formats et tailles.

En utilisant ces étapes, vous pouvez développer un parseur pour le traitement automatisé de formats de documents tels que PDF, DOC, DOCX et DJVU.

exemple

Voici un exemple de code Python pour extraire le texte d'un fichier PDF à l'aide de la bibliothèque PyPDF2 :

pythonCopy code

import PyPDF2

# Ouverture du fichier PDF

with open('exemple.pdf', 'rb') as fichier:

# Instanciation de l'objet PdfFileReader

pdf\_reader = PyPDF2.PdfFileReader(fichier)

# Boucle sur toutes les pages du PDF

for num\_page in range(pdf\_reader.getNumPages()):

# Récupération du contenu textuel de la page

page = pdf\_reader.getPage(num\_page)

contenu = page.extractText()

# Affichage du contenu textuel de la page

print(contenu)

Voici un exemple de code Python pour extraire le texte d'un fichier DOCX à l'aide de la bibliothèque python-docx :

pythonCopy code

import docx

# Ouverture du fichier DOCX

document = docx.Document('exemple.docx')

# Boucle sur tous les paragraphes du document

for paragraphe in document.paragraphs:

# Récupération du contenu textuel du paragraphe

contenu = paragraphe.text

# Affichage du contenu textuel du paragraphe

print(contenu)

Voici un exemple de code Python pour extraire le texte d'un fichier DJVU à l'aide de la bibliothèque PyDjVuLibre :

pythonCopy code

import djvu

# Ouverture du fichier DJVU

document = djvu.decode('exemple.djvu')

# Boucle sur toutes les pages du document

for num\_page in range(document.get\_pages()):

# Récupération du contenu textuel de la page

page = document.get\_page(num\_page)

contenu = page.get\_text()

# Affichage du contenu textuel de la page

print(contenu)



suite

Il est à noter que ces exemples de code ne sont que des exemples simples pour illustrer comment extraire du texte à partir de différents formats de documents. En pratique, le traitement de ces formats de documents peut être beaucoup plus complexe en fonction des besoins spécifiques de l'application.

De plus, il existe de nombreuses autres bibliothèques et outils qui peuvent être utilisés pour extraire du texte à partir de différents formats de documents. Par exemple, pour le traitement des fichiers PDF, d'autres bibliothèques telles que pdfminer et pdfplumber peuvent être utilisées.

Enfin, il est important de noter que l'extraction de texte à partir de formats de documents peut être limitée par la qualité et la structure du document source. Des documents mal structurés, des images contenant du texte et d'autres facteurs peuvent rendre l'extraction de texte plus difficile voire impossible. Il est donc important de prendre en compte ces facteurs lors de la conception d'une application de traitement de documents.



exemple pour traitement de document

Voici un exemple de code Python utilisant la bibliothèque Tika pour extraire du texte et des métadonnées à partir de différents formats de documents :

pythonCopy code

import tika

from tika import parser

# Initialisation de Tika

tika.initVM()

# Liste des chemins vers les fichiers à traiter

chemins\_fichiers = ['exemple.pdf', 'exemple.docx', 'exemple.djvu']

# Boucle sur les fichiers

for chemin\_fichier in chemins\_fichiers:

# Analyse du fichier avec Tika

contenu\_fichier = parser.from\_file(chemin\_fichier)

# Récupération du contenu textuel

contenu\_textuel = contenu\_fichier['content']

# Récupération des métadonnées

metadonnees = contenu\_fichier['metadata']

# Affichage du contenu textuel et des métadonnées

print(f"Contenu textuel de {chemin\_fichier}:")

print(contenu\_textuel)

print("Métadonnées:")

for cle, valeur in metadonnees.items():

print(f"{cle}: {valeur}")

Ce code utilise la bibliothèque Tika pour analyser les fichiers PDF, DOCX et DJVU. La méthode parser.from\_file() est utilisée pour extraire du texte et des métadonnées à partir de chaque fichier. Le contenu textuel et les métadonnées sont ensuite affichés à l'écran.

Il est important de noter que la bibliothèque Tika peut traiter de nombreux autres formats de documents, tels que les fichiers HTML, XML, RTF, etc. En outre, Tika fournit des fonctionnalités avancées telles que la détection de la langue, l'extraction de la structure du document, la reconnaissance optique de caractères (OCR), etc.