Slick Handler

PERSONALIZED BOOKMARK ASSISTANT

Automated Classification of Bookmarks

INDEX

INTRODUCTION

PROBLEM DEFINITION

- Users keep track of the valuable Web sites they have visited and where they would like to return, which is done using a personal list of URLs known as **bookmarks.**
- Well organized bookmarks facilitate user's work and constitute his or her personal information space.
- The current Web browsers offer very limited functionality for the bookmark management, mainly only **manual sorting bookmarks** into **manually created folders.**

GOAL

- To Design ***personal bookmark assistant***, which organizes bookmarks in an ***automated way***
- Bookmark's organizing will be adaptive and incremental, enabling smooth additions of new bookmarks to the recent structure.
- To make the system more ***personal*** and ***assistant***
- System should be able to recommend to a user other Web hyperlinks, which are similar to the already organized bookmarks
- The bookmark personal assistant will classify the bookmarks in the domains:

NEWS	BLOGS	ENTERTAINMENT	SONGS	EDUCATIONAL	SPORTS	OTHERS

which are popular among web surfers.

OBJECTIVES

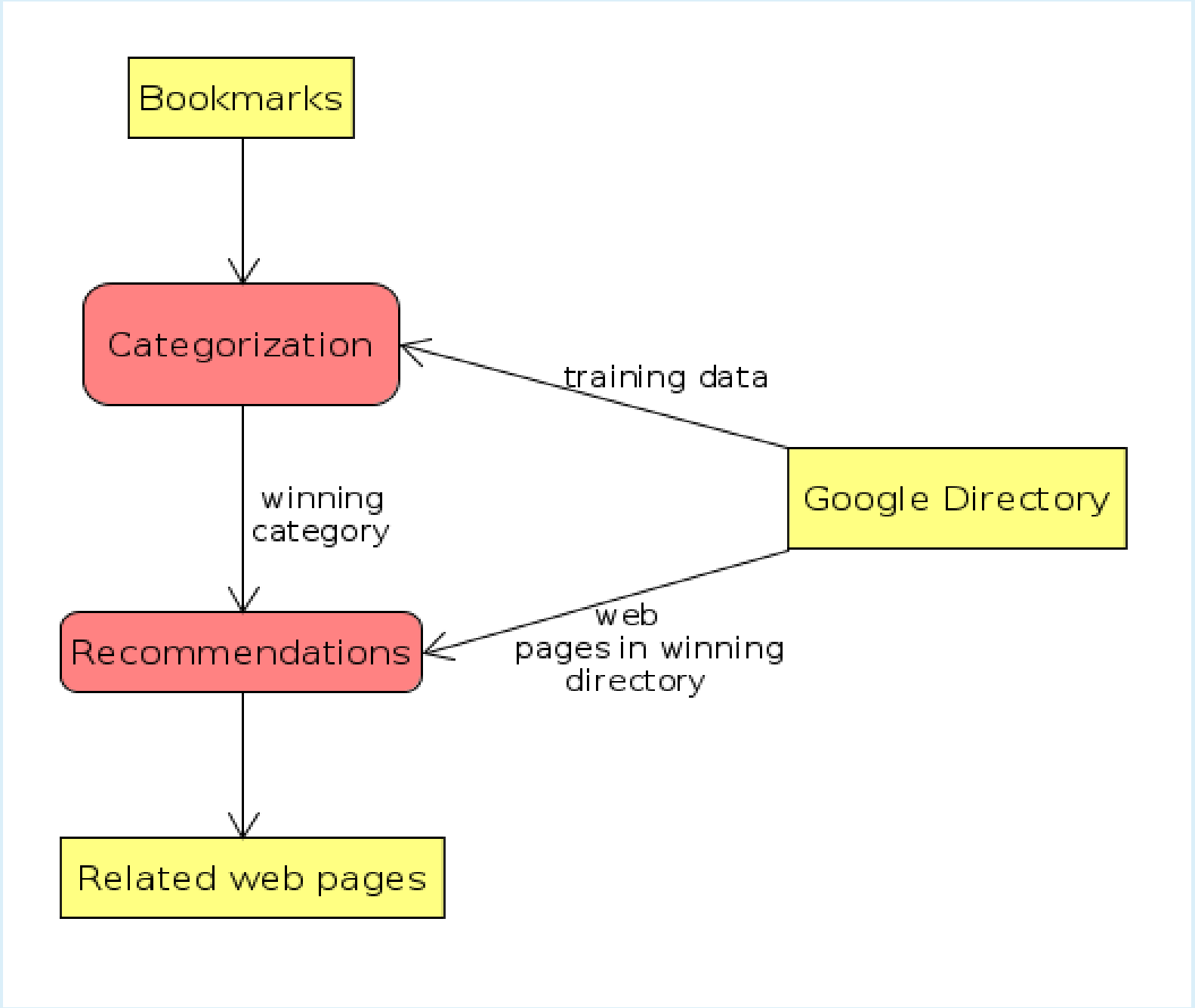
- To develop a personalized bookmark assistant
- To classify the URLS based on the domains : news, blogs, videos, songs, educational and sports
- To be able to recommend to a user other Web hyperlinks, which are similar to the already organized bookmarks
- To make the experience more user-friendly

ARCHITECTURE

TECHNICAL DETAILS

- *Techniques Used-*
  - ❑ Supervised Machine Learning Approach – Textual Classification
  - ❑ Natural Language Processing (NLP)
  - ❑ Web Crawling
- *Languages Used* : Python, HTML, CSS, JavaScript, Json
- *Library used* : Natural Language ToolKit (NLTK)
- *Dataset Used* : DMOZ (Google Open directory project)
- *Memory Used* : 332 MB by dataset

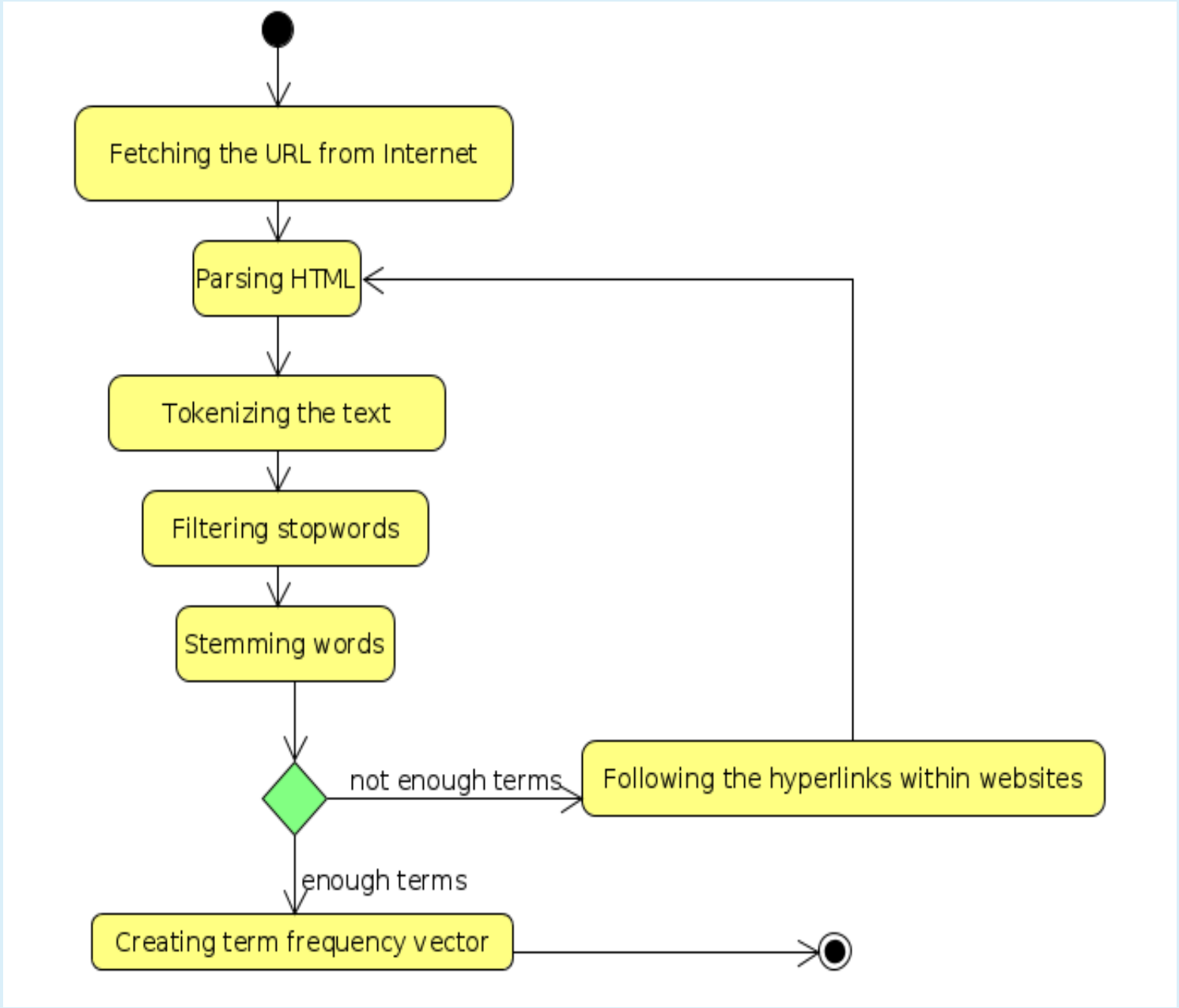
TRAINING OF DATA


Bookmarks
Categorization
training data
Google Directory
winning
category
web
pages in winning
directory
Recommendations
Related web pages

TRAINING DATA AND PRE-CLASSIFICATION

- Dataset used : DMOZ Google Open Directory Project
- DMOZ contains URL's of various website in an categorical order
- Domain For our Project is:

NEWS	BLOGS	ENTERTAINMENT	SONGS	EDUCATIONAL	SPORTS	OTHERS



Steps in Pre-Classification

Web Document Indexing

Step 1: HTML parsing

HTML parser omits useless parts of the code and extracts only the following:

- ❑ Title
- ❑ Meta Keyword Tag
- ❑ Meta Description Tag
- ❑ Normal Text
- ❑ Links to other pages which are stored at same serve

Web Document Indexing

Step 2 : Tokenisation

- ❑ Extracting words from the text
- ❑ For This we will use nltk.tokenizer

Input: Friends, Romans, Countrymen, lend me your ears;

Output:

Friends	Romans	Countrymen	lend	me	your	ears

Web Document Indexing

Step 3: Stop word deletion

- Words that are not beneficial for information retrieval are removed
- General English stop words are removed like “the”, “of” etc.
- For This we will use nltk.stopwords

Sample text with Stop Words	Without Stop Words
GeeksforGeeks – A Computer Science Portal for Geeks	GeeksforGeeks , Computer Science, Portal ,Geeks
Can listening be exhausting?	Listening, Exhausting
I like reading, so I read	Like, Reading, read

Web Document Indexing

Step 4: Stemming

- A program that reduces word forms to a canonical form
- This is done by removal of the various suffixes like -ed, -ing, -ion, -ions etc. to leave the single term.
- For this we used porter stemmer
- For instance:

“connect”, “connected”, “connection”, “connections” can be stemmed to the single term “connect”.

Web Document Indexing

Step 5: Term Frequency- Inverse Document Frequency (TF-IDF)

- *TF:*
- Measures how frequently a term occurs in a document.
- TF(t) = (Number of times term t appears in a document) / (Total number of terms in the document).

- *IDF:*
- Measures how important a term is.
- IDF(t) = log_e(Total number of documents / Number of documents with term t in it).

TESTING DATA

For Testing of data :

- **Step 1:** Fetch URL
- **Step 2: Web Document Indexing** similar to training set
- **Step 3: Centroid based classification** model to determine the category
- **Step 4:** Updating the dataset with new word vectors from test data

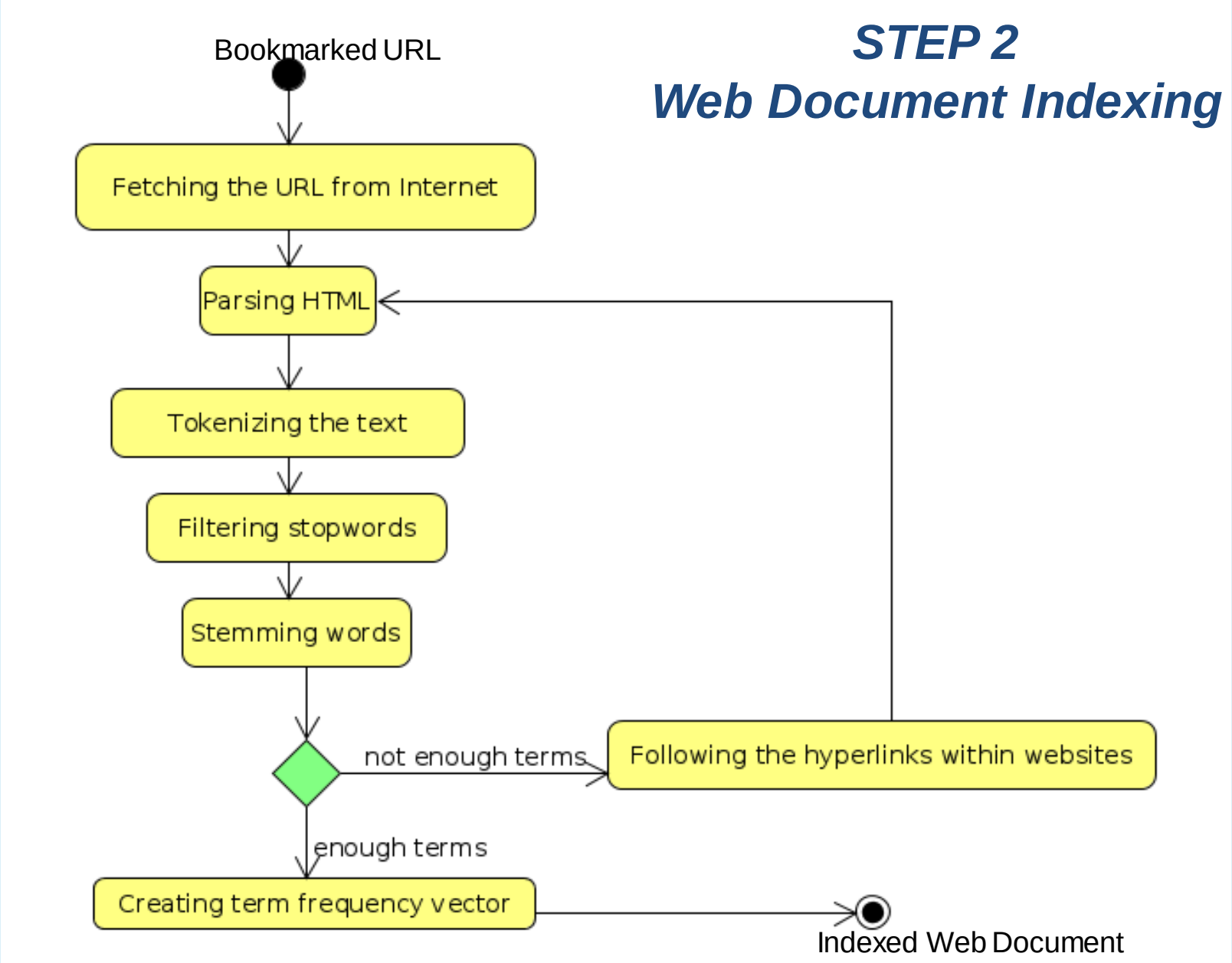


Fig. Activity diagram of a web document indexing

Step 3: CENTROID-BASED DOCUMENT CLASSIFICATION

- We used Rocchio's centroid algorithm, based on the vector space model
- Document is represented as vectors in a term space.
- For each category a centroid vector is created as a representative.
- Given k categories in the training set, this leads to k centroid vectors

c1, c2, ... ck,

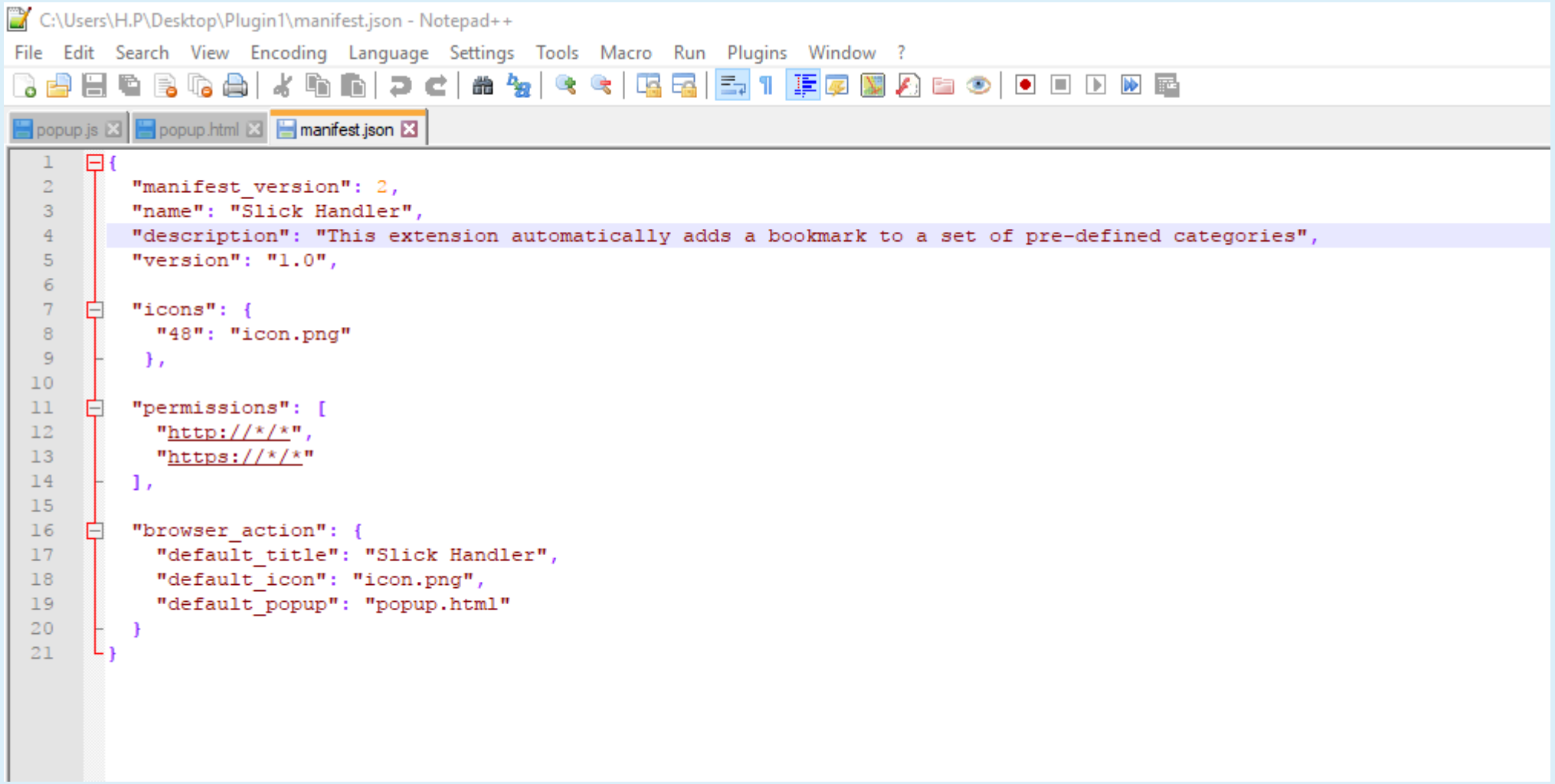
where each ci is the centroid for ith category, the category of new document vector is as follows:

$$\arg \max_{j=1,\ldots,k} (\cos(\vec{x}, \vec{c_j}))$$

CRITERIA FOR CATEGORISATION

- The classification is considered as unsuccessful if, score for tested document is not higher than the ***THRESHOLD***.
- The tested document has to be similar to one of the training documents at least in 10%
- Else the classification will be to the other folder or system will ask for the creation of new folder.

IMPLEMENTATION



EXTENSION CODE:
File 1: manifest.json

C:\Users\H.P\Desktop\Plugin1\popup.js - Notepad++

File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?

popup.js | popup.html | manifest.json

```
1  function myAlert(){
2      alert('Added to <floder name> successfully');
3  }
4
5  document.addEventListener('DOMContentLoaded', function () {
6      document.getElementById('alertButton').addEventListener('click', myAlert);
7  });
```

File 2: Popup.js

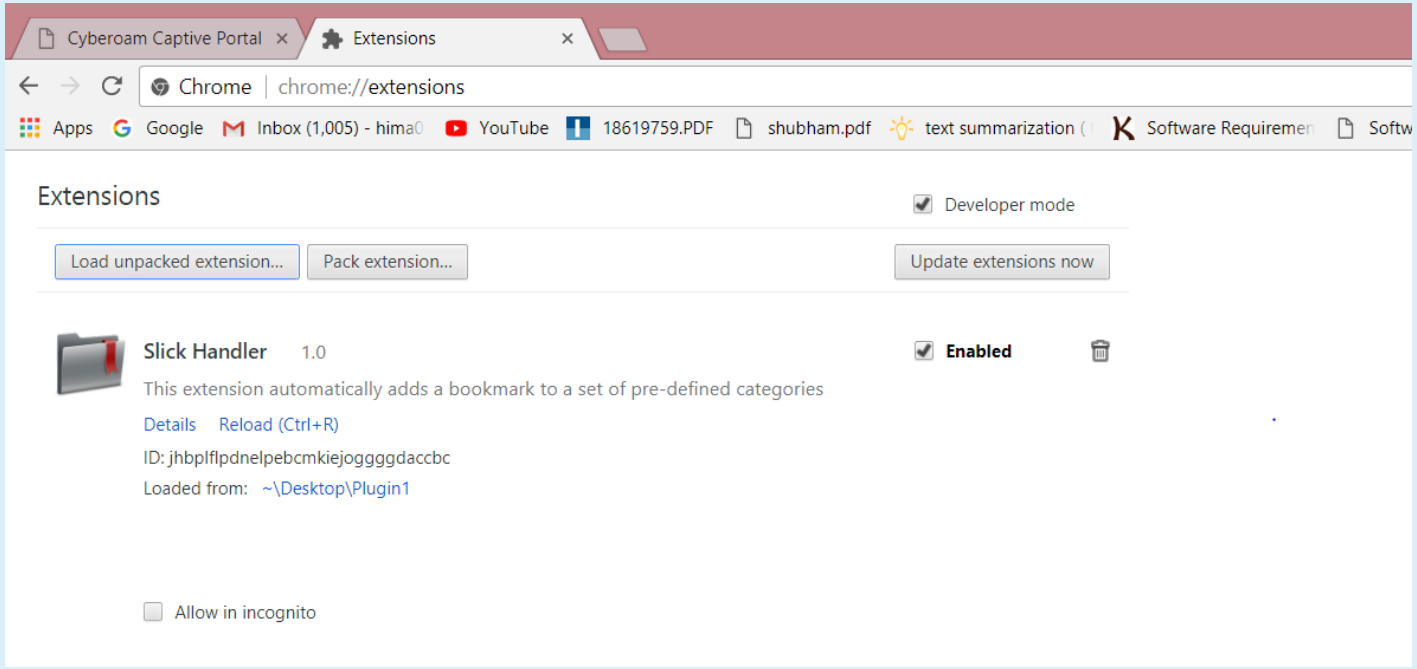
C:\Users\H.P\Desktop\Plugin1\popup.html - Notepad++

File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?

popup.js | popup.html | manifest.json

```
1  <html>
2      <head>
3          <title>Slick Handler</title>
4          <script language='javascript' src='popup.js'></script>
5      </head>
6      <body>
7          <form name='testForm'>
8              <input type='button' id='alertButton' value='Add To Bookmarks'>
9              <input style="margin-top: 16px" type='button' id='alertButton' value='View Bookmarks'>
10         </form>
11     </body>
12 </html>
```

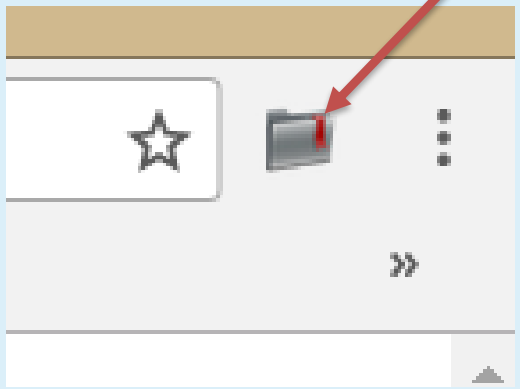
File 3: Popup.html

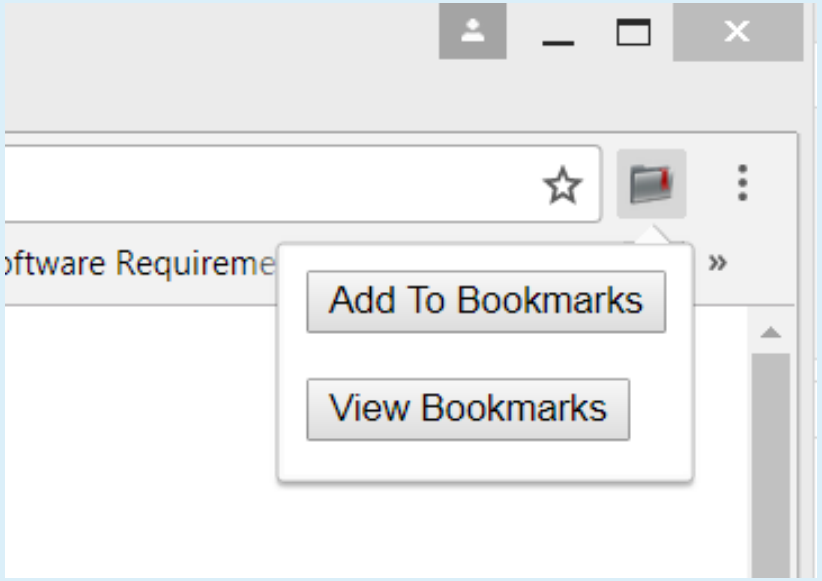


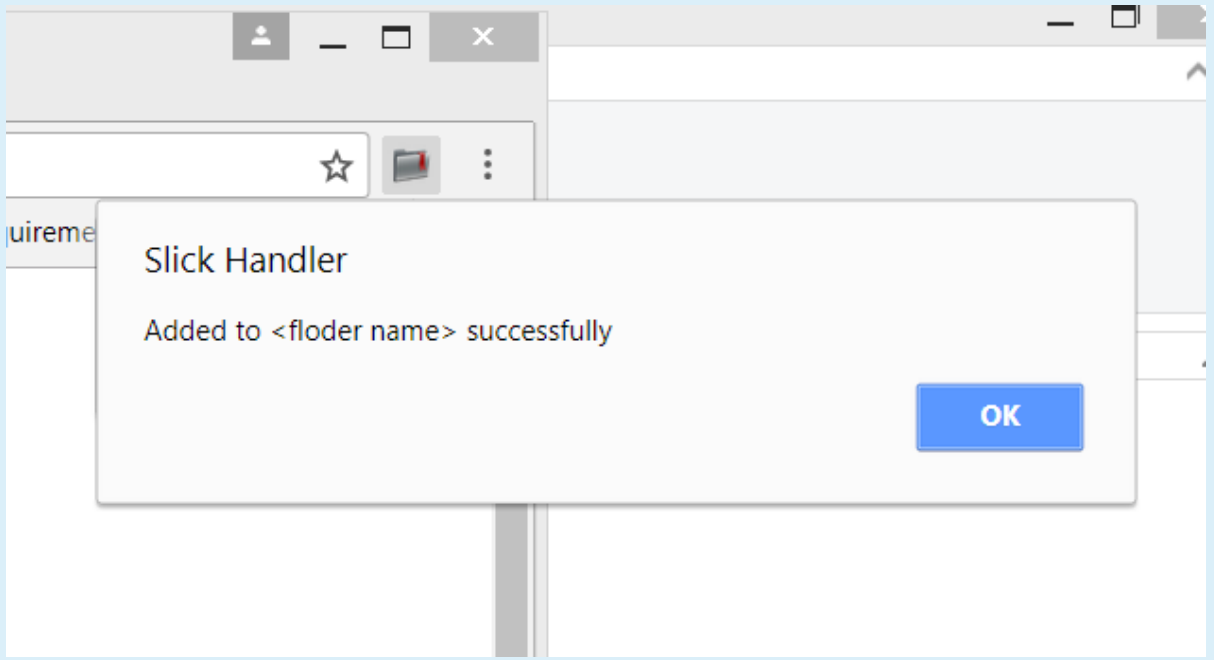
Browser Extension :

“ Slick Handler “

Slick Handler

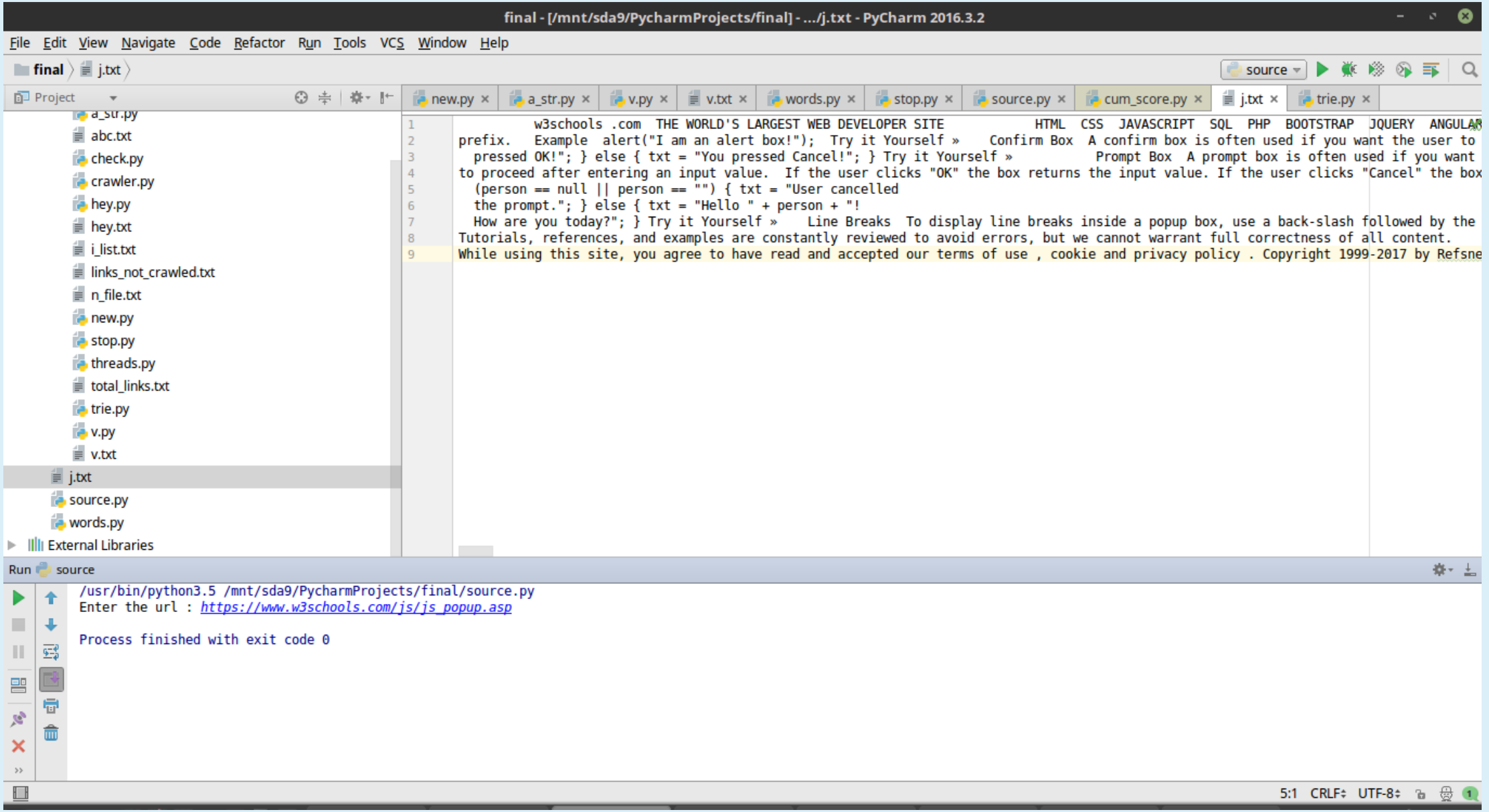



ftware Requireme
Add To Bookmarks
View Bookmarks


uireme
Slick Handler
Added to <floder name> successfully
OK

Fetching The URL : Web Crawler

```
from __future__ import division
from bs4 import BeautifulSoup
from bs4.element import Comment
'''from urllib.request import urlopen'''
import urllib









def tag_visible(element):
    if element.parent.name in ['style', 'script', 'head', 'title', 'meta', '[document]']:
        return False
    if isinstance(element, Comment):
        return False
    return True


def text_from_html(body):
    soup = BeautifulSoup(body, 'html.parser')
    texts = soup.findAll(text=True)
    visible_texts = filter(tag_visible, texts)
    return u" ".join(t.strip() for t in visible_texts)

url = input("Enter the url : ")
html = urlopen(url).read()
f = open('file1.txt', 'w')
text = text_from_html(html)
f.write(text)
f.close()
```



Text Extracted from a website through web crawler

```
from __future__ import division
from nltk.corpus import stopwords
from string import punctuation
from nltk.stem import PorterStemmer
from nltk import word_tokenize
from nltk import sent_tokenize
from operator import truediv
from sklearn.feature_extraction.text import TfidfVectorizer
import numpy as np
import sys
```

Python code for training and testing data

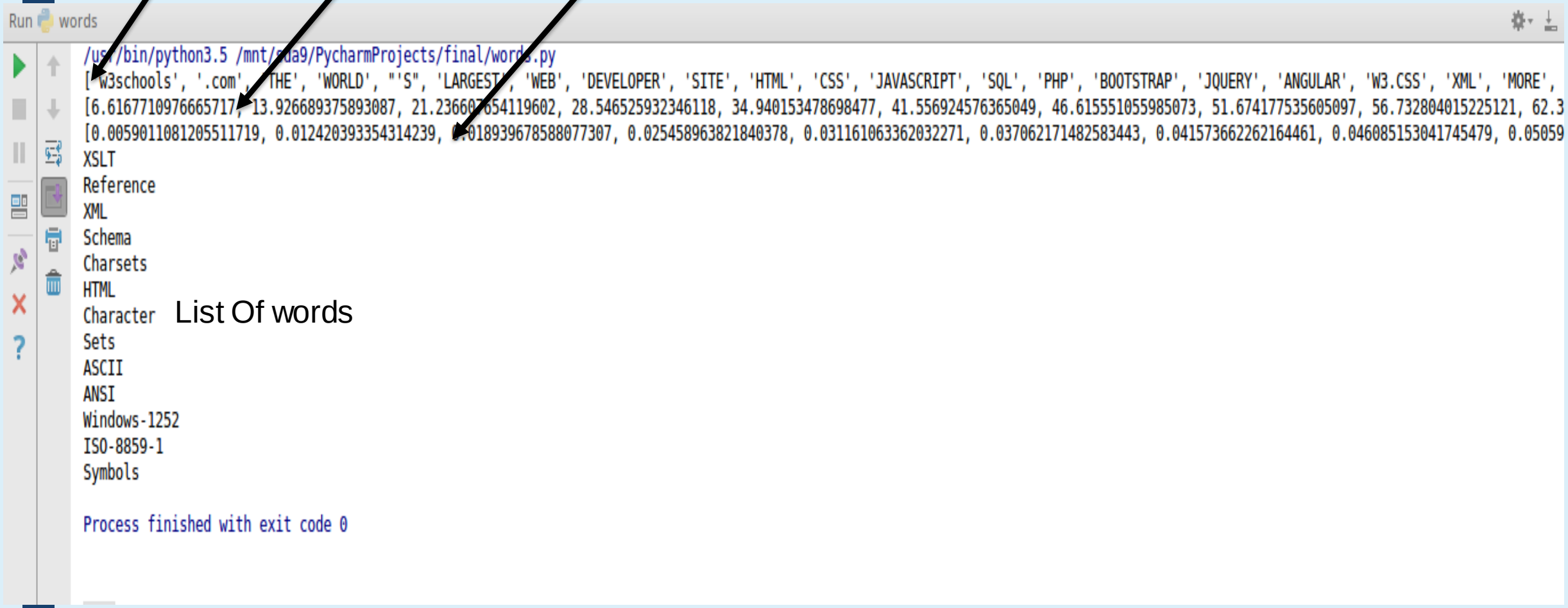
Python code for training and testing data

```
score=[]

for i in stem_words:
    a=0
    if i in dict.keys():
            a=a+dict[i]
            score.append(a)


score1=[float(v) for v in score]
max=score1[0]
for i in score1:
    if i>max:
        max=i
j=0
for i in score1:
    score1[j]=i/max
    j=j+1



word_dict={}
for i,j in zip(score1,word):
    word_dict[i]=j

sort_tot=sorted(score1)

final_words=[]
for i in range(n-8,n):
    final_words.append(word_dict[sort_tot[i]])

for i in final_words:
    print(i)s
```

Normalized Score

Score

Tokenized words



List Of words

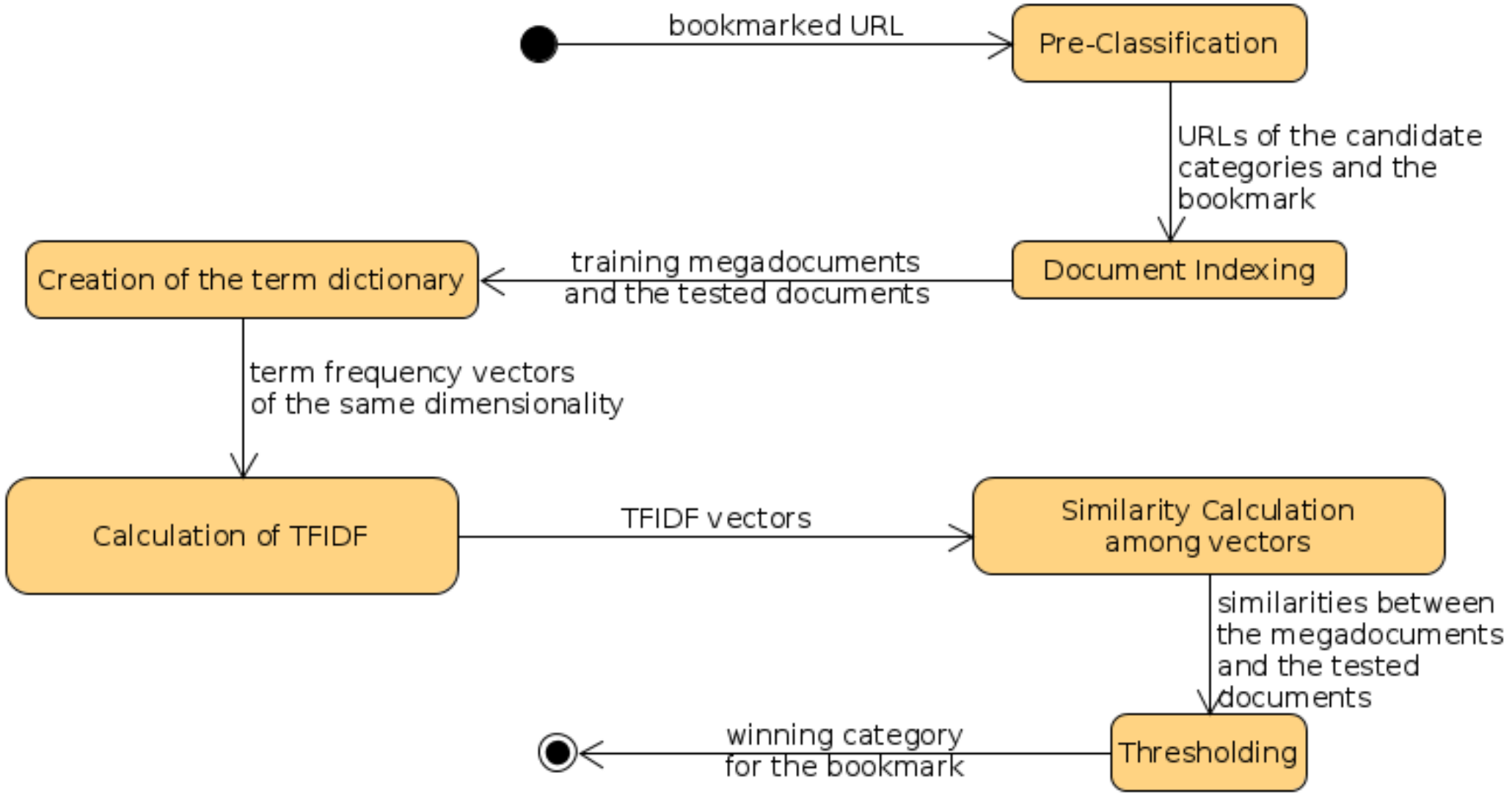


Fig. Overall System View

PROPOSED DELIVERABLES

- We proposed to deliver to a browser extension which will categorises the bookmarks
- The extension will also recommend the related web pages for the bookmarked sites
- The extension will also allow user to change the destination folder as per his requirement that is the extension will work on the automation as well as will work manually

SCOPE

- Our extension will be able to add a bookmark to the best fitting category from the pre-defined set of categories, which are

NEWS	BLOGS	ENTERTAINMENT	SONGS	EDUCATIONAL	SPORTS	OTHERS

- Users will be able to manage their bookmarks more easily and efficiently with the automation of bookmark manager

- For future aspects, we wish to add the search bar as well as wish to extend our domains further with the help of unsupervised machine learning