```
from flask import Flask, render_template, request, jsonify
from urlRetrieval_Tfidf import sentence_similarity_query

app = Flask(__name__)
#app.debug=True


@app.route("/")
def home():
    return render_template("index.html")

@app.route('/', methods=['POST'])
def hello():
    UserInput = request.form['msg']
    processed_text = sentence_similarity_query(UserInput.lower())
    return processed_text


@app.route("/", methods=['GET'])
def get_bot_response():
    userText = request.args.get('msg')
    print(userText)


@app.route('/', methods=['POST'])
def my_form_post():
    text = request.form['text']
    processed_text = text.upper()
    return processed_text


if __name__ == "__main__":
    app.run(debug=True)
```

URLRetreival_POS.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Mon Feb 12 22:00:56 2018

@author: pankaj
```

```
"""

from nltk import word_tokenize, pos_tag
from nltk.corpus import wordnet as wn
import re


df = pd.read_csv("urls.csv", header=0)

urlList = df.URLS.tolist()

def penn_to_wn(tag):
    """ Convert between a Penn Treebank tag to a simplified Wordnet tag """
    if tag.startswith('N'):
        return 'n'

    if tag.startswith('V'):
        return 'v'

    if tag.startswith('J'):
        return 'a'

    if tag.startswith('R'):
        return 'r'

    return None

def tagged_to_synset(word, tag):
    wn_tag = penn_to_wn(tag)
    if wn_tag is None:
        return None

    try:
        return wn.synsets(word, wn_tag)[0]
    except:
        return None

def sentence_similarity(sentence1, sentence2):
    """ compute the sentence similarity using Wordnet """
    # Tokenize and tag
    sentence1 = pos_tag(word_tokenize(sentence1))
    sentence2 = pos_tag(word_tokenize(sentence2))
    # Get the synsets for the tagged words
    synsets1 = [tagged_to_synset(*tagged_word) for tagged_word in sentence1]
    synsets2 = [tagged_to_synset(*tagged_word) for tagged_word in sentence2]
    # Filter out the Nones
    synsets1 = [ss for ss in synsets1 if ss]
    synsets2 = [ss for ss in synsets2 if ss]
    score, count = 0.0, 0.001

    # For each word in the first sentence
    for synset in synsets1:
        # Get the similarity value of the most similar word in the other sentence
        # print [synset.path_similarity(ss) for ss in synsets2]
        try:
            best_score = max([synset.path_similarity(ss) for ss in synsets2])
        except:
            best_score = 0.0
        # Check that the similarity could have been computed
        if best_score is not None:
            score += best_score
            count += 1

    # Average the values
    score /= count
    return score


def symmetric_sentence_similarity(sentence1, sentence2):
    """ compute the symmetric sentence similarity using Wordnet """
    #print (sentence1,sentence2)
    return (sentence_similarity(sentence1, sentence2) + sentence_similarity(sentence2,
sentence1)) / 2


def sentence_similarity_query(userInput):
    userInput = " ".join(re.compile('\w+').findall(userInput.lower()))
    print(userInput)
    score_list=[]
    for url in urlList:
        print(url)
        url = " ".join(re.compile('\w+').findall(url.lower()))
        score_list.append(symmetric_sentence_similarity(sentence1,url))
        print(score_list)
    best_match = urlList[score_list.index(min(score_list))]
    return best_match
```

```
#sentence1="https://www.odyssey.com/dal/peoplesoft"
sentence2="I want a superdatascience features"
#sentence1 = pos_tag(word_tokenize(sentence1))
#" ".join(re.compile('\w+').findall(sentence1))

sentence_similarity_query(sentence2)
```

URLRetrieval_Tfidf.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Tue Feb 13 14:37:42 2018

@author: pankaj
"""

#!/usr/bin/env python3
# -*- coding: utf-8 -*-

import pandas as pd
import re
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity

df = pd.read_csv("urls.csv", header=0)

urlList = df.URLS.tolist()


def sentence_similarity(sentence1, sentence2):
    """ compute the cosine similarity using Tfidf Vectorizer """
    # Tokenize and tag
    finalText = [sentence1 , sentence2]

    vectorizer = TfidfVectorizer()
    doc_vector = vectorizer.fit_transform(finalText)

    df = pd.DataFrame(doc_vector.toarray().transpose(), index=vectorizer.get_feature_names(),
columns = ['userInput', 'URL'])

    txt1 = df['userInput'].values.reshape(1, -1)
    txt2 = df['URL'].values.reshape(1, -1)

    return cosine_similarity(txt1, txt2)


def sentence_similarity_query(userInput):
    userInput = " ".join(re.compile('\w+').findall(userInput.lower()))
    print(userInput)
    score_list=[]
    for url in urlList:
        url = " ".join(re.compile('\w+').findall(url.lower()))
        score_list.append(sentence_similarity(userInput,url))
    best_match = urlList[score_list.index(max(score_list))]
    return best_match



#sentence1="https://www.odyssey.com/dal/peoplesoft"
sentence2="star technical document"
#sentence1 = pos_tag(word_tokenize(sentence1))
#" ".join(re.compile('\w+').findall(sentence1))

sentence_similarity_query(sentence2)
```

index

```
<!DOCTYPE html>
<html>
 <head>
    <title>Home</title>
    <link rel="stylesheet" type="text/css" href="/static/styles.css">
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js"></script>
 </head>
  <body>
    <h1>Document Search Engine</h1>
    <h3>Chatbot to find document links in Odyssey / WIKI / Confluence.</h3>
      <form action="/" method="post">
      UserInput: <input type="text" name="msg" placeholder="Message"/>
      <input type="submit" name="form"value="Submit"/>
    </form>
  </body>
</html>
```

Rnn.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Mon Feb 12 22:31:14 2018

@author: pankaj
"""

import numpy as np
import tensorflow as tf
import re
import time



########## PART 1 - DATA PREPROCESSING ##########



# Importing the dataset
lines = open('movie_lines.txt', encoding = 'utf-8', errors = 'ignore').read().split('\n')
conversations = open('movie_conversations.txt', encoding = 'utf-8', errors =
'ignore').read().split('\n')

# Creating a dictionary that maps each line and its id
id2line = {}
for line in lines:
    _line = line.split(' +++$+++ ')
    if len(_line) == 5:
        id2line[_line[0]] = _line[4]

# Creating a list of all of the conversations
conversations_ids = []
for conversation in conversations[:-1]:
    _conversation = conversation.split(' +++$+++ ')[-1][1:-1].replace("'", "").replace(" ", "")
    conversations_ids.append(_conversation.split(','))

# Getting separately the questions and the answers
questions = []
answers = []
for conversation in conversations_ids:
    for i in range(len(conversation) - 1):
        questions.append(id2line[conversation[i]])
        answers.append(id2line[conversation[i+1]])

# Doing a first cleaning of the texts
def clean_text(text):
    text = text.lower()
    text = re.sub(r"i'm", "i am", text)
    text = re.sub(r"he's", "he is", text)
    text = re.sub(r"she's", "she is", text)
    text = re.sub(r"that's", "that is", text)
    text = re.sub(r"what's", "what is", text)
    text = re.sub(r"where's", "where is", text)
    text = re.sub(r"\'ll", " will", text)
    text = re.sub(r"\'ve", " have", text)
    text = re.sub(r"\'re", " are", text)
    text = re.sub(r"\'d", " would", text)
    text = re.sub(r"won't", "will not", text)
    text = re.sub(r"can't", "cannot", text)
    text = re.sub(r"[-()\"#/@;:<>{}+=~|.?,]", "", text)
    return text

# Cleaning the questions
clean_questions = []
for question in questions:
    clean_questions.append(clean_text(question))

# Cleaning the answers
clean_answers = []
for answer in answers:
    clean_answers.append(clean_text(answer))

# Creating a dictionary that maps each word to its number of occurrences
word2count = {}
for question in clean_questions:
    for word in question.split():
        if word not in word2count:
            word2count[word] = 1
        else:
            word2count[word] += 1
for answer in clean_answers:
    for word in answer.split():
        if word not in word2count:
            word2count[word] = 1
        else:
            word2count[word] += 1
```

```
# Creating two dictionaries that map the questions words and the answers words to a unique integer
threshold_questions = 20
questionswords2int = {}
word_number = 0
for word, count in word2count.items():
    if count >= threshold_questions:
        questionswords2int[word] = word_number
        word_number += 1
threshold_answers = 20
answerswords2int = {}
word_number = 0
for word, count in word2count.items():
    if count >= threshold_answers:
        answerswords2int[word] = word_number
        word_number += 1

# Adding the last tokens to these two dictionaries
tokens = ['<PAD>', '<EOS>', '<OUT>', '<SOS>']
for token in tokens:
    questionswords2int[token] = len(questionswords2int) + 1
for token in tokens:
    answerswords2int[token] = len(answerswords2int) + 1

# Creating the inverse dictionary of the answerswords2int dictionary
answersints2word = {w_i: w for w, w_i in answerswords2int.items()}

# Adding the End Of String token to the end of every answer
for i in range(len(clean_answers)):
    clean_answers[i] += ' <EOS>'

# Translating all the questions and the answers into integers
# and Replacing all the words that were filtered out by <OUT>
questions_into_int = []
for question in clean_questions:
    ints = []
    for word in question.split():
        if word not in questionswords2int:
            ints.append(questionswords2int['<OUT>'])
        else:
            ints.append(questionswords2int[word])
    questions_into_int.append(ints)
answers_into_int = []
```

```
for answer in clean_answers:
    ints = []
    for word in answer.split():
        if word not in answerswords2int:
            ints.append(answerswords2int['<OUT>'])
        else:
            ints.append(answerswords2int[word])
    answers_into_int.append(ints)

# Sorting questions and answers by the length of questions
sorted_clean_questions = []
sorted_clean_answers = []
for length in range(1, 25 + 1):
    for i in enumerate(questions_into_int):
        if len(i[1]) == length:
            sorted_clean_questions.append(questions_into_int[i[0]])
            sorted_clean_answers.append(answers_into_int[i[0]])



########## PART 2 - BUILDING THE SEQ2SEQ MODEL ##########



# Creating placeholders for the inputs and the targets
def model_inputs():
    inputs = tf.placeholder(tf.int32, [None, None], name = 'input')
    targets = tf.placeholder(tf.int32, [None, None], name = 'target')
    lr = tf.placeholder(tf.float32, name = 'learning_rate')
    keep_prob = tf.placeholder(tf.float32, name = 'keep_prob')
    return inputs, targets, lr, keep_prob

# Preprocessing the targets
def preprocess_targets(targets, word2int, batch_size):
    left_side = tf.fill([batch_size, 1], word2int['<SOS>'])
    right_side = tf.strided_slice(targets, [0,0], [batch_size, -1], [1,1])
    preprocessed_targets = tf.concat([left_side, right_side], 1)
    return preprocessed_targets

# Creating the Encoder RNN
def encoder_rnn(rnn_inputs, rnn_size, num_layers, keep_prob, sequence_length):
    lstm = tf.contrib.rnn.BasicLSTMCell(rnn_size)
    lstm_dropout = tf.contrib.rnn.DropoutWrapper(lstm, input_keep_prob = keep_prob)
    encoder_cell = tf.contrib.rnn.MultiRNNCell([lstm_dropout] * num_layers)
    encoder_output, encoder_state = tf.nn.bidirectional_dynamic_rnn(cell_fw = encoder_cell,
                                  cell_bw = encoder_cell,
                                  sequence_length = sequence_length,
                                  inputs = rnn_inputs,
                                  dtype = tf.float32)
    return encoder_state

# Decoding the training set
def decode_training_set(encoder_state, decoder_cell, decoder_embedded_input,
sequence_length, decoding_scope, output_function, keep_prob, batch_size):
    attention_states = tf.zeros([batch_size, 1, decoder_cell.output_size])
    attention_keys, attention_values, attention_score_function, attention_construct_function =
tf.contrib.seq2seq.prepare_attention(attention_states, attention_option = "bahdanau",
num_units = decoder_cell.output_size)
    training_decoder_function =
tf.contrib.seq2seq.attention_decoder_fn_train(encoder_state[0],
                                       attention_keys,
                                       attention_values,
                                       attention_score_function,
                                       attention_construct_function,
                                       name = "attn_dec_train")
    decoder_output, decoder_final_state, decoder_final_context_state =
tf.contrib.seq2seq.dynamic_rnn_decoder(decoder_cell,
                                                        training_decoder_function,
                                                        decoder_embedded_input,
                                                        sequence_length,
                                                        scope = decoding_scope)
    decoder_output_dropout = tf.nn.dropout(decoder_output, keep_prob)
    return output_function(decoder_output_dropout)

# Decoding the test/validation set
def decode_test_set(encoder_state, decoder_cell, decoder_embeddings_matrix, sos_id,
eos_id, maximum_length, num_words, decoding_scope, output_function, keep_prob,
batch_size):
    attention_states = tf.zeros([batch_size, 1, decoder_cell.output_size])
    attention_keys, attention_values, attention_score_function, attention_construct_function =
tf.contrib.seq2seq.prepare_attention(attention_states, attention_option = "bahdanau",
num_units = decoder_cell.output_size)
    test_decoder_function =
tf.contrib.seq2seq.attention_decoder_fn_inference(output_function,
                                       encoder_state[0],
                                       attention_keys,
                                       attention_values,
                                       attention_score_function,
                                                                              attention_construct_function,
                                                                              decoder_embeddings_matrix,
                                                                              sos_id,
                                                                              eos_id,
                                                                              maximum_length,
                                                                              num_words,
                                                                              name = "attn_dec_inf")
    test_predictions, decoder_final_state, decoder_final_context_state =
tf.contrib.seq2seq.dynamic_rnn_decoder(decoder_cell,
                                                                                                                test_decoder_function,
                                                                                                                scope = decoding_scope)
    return test_predictions

# Creating the Decoder RNN
def decoder_rnn(decoder_embedded_input, decoder_embeddings_matrix, encoder_state,
num_words, sequence_length, rnn_size, num_layers, word2int, keep_prob, batch_size):
    with tf.variable_scope("decoding") as decoding_scope:
        lstm = tf.contrib.rnn.BasicLSTMCell(rnn_size)
        lstm_dropout = tf.contrib.rnn.DropoutWrapper(lstm, input_keep_prob = keep_prob)
        decoder_cell = tf.contrib.rnn.MultiRNNCell([lstm_dropout] * num_layers)
        weights = tf.truncated_normal_initializer(stddev = 0.1)
        biases = tf.zeros_initializer()
        output_function = lambda x: tf.contrib.layers.fully_connected(x,
                                                                      num_words,
                                                                      None,
                                                                      scope = decoding_scope,
                                                                      weights_initializer = weights,
                                                                      biases_initializer = biases)
        training_predictions = decode_training_set(encoder_state,
                                                   decoder_cell,
                                                   decoder_embedded_input,
                                                   sequence_length,
                                                   decoding_scope,
                                                   output_function,
                                                   keep_prob,
                                                   batch_size)
        decoding_scope.reuse_variables()
        test_predictions = decode_test_set(encoder_state,
                                           decoder_cell,
                                           decoder_embeddings_matrix,
                                           word2int['<SOS>'],
                                           word2int['<EOS>'],
                                           sequence_length - 1,
                                           num_words,
                                           decoding_scope,
                                           output_function,
                                           keep_prob,
                                           batch_size)
    return training_predictions, test_predictions

# Building the seq2seq model
def seq2seq_model(inputs, targets, keep_prob, batch_size, sequence_length,
answers_num_words, questions_num_words, encoder_embedding_size,
decoder_embedding_size, rnn_size, num_layers, questionswords2int):
    encoder_embedded_input = tf.contrib.layers.embed_sequence(inputs,
                                                              answers_num_words + 1,
                                                              encoder_embedding_size,
                                                              initializer = tf.random_uniform_initializer(0, 1))
    encoder_state = encoder_rnn(encoder_embedded_input, rnn_size, num_layers, keep_prob,
sequence_length)
    preprocessed_targets = preprocess_targets(targets, questionswords2int, batch_size)
    decoder_embeddings_matrix = tf.Variable(tf.random_uniform([questions_num_words + 1,
decoder_embedding_size], 0, 1))
    decoder_embedded_input = tf.nn.embedding_lookup(decoder_embeddings_matrix,
preprocessed_targets)
    training_predictions, test_predictions = decoder_rnn(decoder_embedded_input,
                                                         decoder_embeddings_matrix,
                                                         encoder_state,
                                                         questions_num_words,
                                                         sequence_length,
                                                         rnn_size,
                                                         num_layers,
                                                         questionswords2int,
                                                         keep_prob,
                                                         batch_size)
    return training_predictions, test_predictions
```