**Legal Document Analyzer using Flask and IBM Watson**

**Introduction:**

The "Legal Document Analyzer" represents a groundbreaking solution that converges advanced technologies with user-centric design. It operates within the IBM Watson Cloud ecosystem and stands as a testament to the synergy of artificial intelligence, machine learning, and specialized legal knowledge. This introduction sets the stage for an in-depth exploration of how this innovative tool revolutionizes the analysis of legal documents, offering swift, accurate insights to legal professionals and individuals seeking to streamline document review, enhance accuracy, and make well-informed decisions in the legal domain.

import sys

import nltk.data

import os

from nltk.corpus import stopwords

from nltk.stem import WordNetLemmatizer

import math

import numpy as np

from nltk import word\_tokenize, pos\_tag, ne\_chunk

from nltk.tree import Tree

from nltk.compat import python\_2\_unicode\_compatible

import re

import operator

df\_vec = {}

doc\_w\_vec = {}

total\_docs = 0

def cal\_df():

global df\_vec

tokenizer = nltk.data.load('tokenizers/punkt/english.pickle')

path = input\_path

files = os.listdir(path)

for f in files:

f\_path = path + "\\" + f

try:

fp = open(f\_path)

except:

print("didn't open")

data = fp.read()

fp.close()

sntncs = tokenizer.tokenize(data)

nor\_stp\_lmt = []

try:

wordnet\_lemmatizer = WordNetLemmatizer()

stop = set(stopwords.words('english'))

for s in sntncs:

s\_nor\_stp\_lmt = ""

s = s.lower()

words = word\_tokenize(s)

for w in words:

if w not in stop:

w = wordnet\_lemmatizer.lemmatize(w)

s\_nor\_stp\_lmt = s\_nor\_stp\_lmt + w + " "

nor\_stp\_lmt.append(s\_nor\_stp\_lmt)

except:

print("wordnet")

unq\_words = {}

for s in nor\_stp\_lmt:

for w in word\_tokenize(s):

if w != ".":

if w not in unq\_words:

unq\_words[w] = 0

for k in unq\_words.keys():

if k in df\_vec:

df\_vec[k] = df\_vec[k] + 1

else:

df\_vec[k] = 1

def cal\_total\_doc():

global total\_docs

path = input\_path

files = os.listdir(path)

total\_docs = len(files)

def get\_continuous\_chunks(text):

chunked = ne\_chunk(pos\_tag(word\_tokenize(text))

continuous\_chunk = []

current\_chunk = []

for i in chunked:

if type(i) == Tree:

current\_chunk.append(" ".join([token for token, pos in i.leaves()]))

elif current\_chunk:

named\_entity = " ".join(current\_chunk)

if named\_entity not in continuous\_chunk:

continuous\_chunk.append(named\_entity)

current\_chunk = []

else:

continue

return continuous\_chunk

legal\_words = []

def read\_legal\_dict():

l\_f = open("dictionary.txt", "r")

for wd in l\_f:

legal\_words.append(wd)

l\_f.close()

def cal\_tf\_Idf():

global legal\_words

global total\_docs

global doc\_w\_vec

global df\_vec

tokenizer = nltk.data.load('tokenizers/punkt/english.pickle')

path = input\_path

files = os.listdir(path)

for f in files:

tf\_idf\_sntnc = {}

f\_path = path + "\\" + f

fp = open(f\_path)

data = fp.read()

fp.close()

sntncs = tokenizer.tokenize(data)

nor\_stp\_lmt = []

stp\_lmt\_cased = []

wordnet\_lemmatizer = WordNetLemmatizer()

stop = set(stopwords.words('english'))

for s in sntncs:

s\_nor\_stp\_lmt = ""

s\_u = s.lower()

words = word\_tokenize(s\_u)

for w in words:

if w not in stop:

w = wordnet\_lemmatizer.lemmatize(w)

s\_nor\_stp\_lmt = s\_nor\_stp\_lmt + w + " "

nor\_stp\_lmt.append(s\_nor\_stp\_lmt)

words = word\_tokenize(s)

case\_sntnc = ""

for w in words:

if w not in stop:

w = wordnet\_lemmatizer.lemmatize(w)

case\_sntnc = case\_sntnc + w + " "

stp\_lmt\_cased.append(case\_sntnc)

tf\_vec = {}

length = 0

for i in range(len(nor\_stp\_lmt):

s = nor\_stp\_lmt[i]

for w in word\_tokenize(s):

if w != ".":

length = length + 1

if w in tf\_vec:

tf\_vec[w] = tf\_vec[w] + 1

else:

tf\_vec[w] = 1

tf\_idf\_doc = {}

for k in tf\_vec.keys():

tf\_vec[k] = float(tf\_vec[k]) / float(length)

tf\_idf\_doc[k] = tf\_vec[k] \* math.log10(float(total\_docs) / float(df\_vec[k])

doc\_w\_vec[fp] = tf\_idf\_doc

tf\_idf\_sntnc = {}

std\_list = []

for i in range(len(nor\_stp\_lmt)):

s = nor\_stp\_lmt[i]

ac\_s = sntncs[i]

sm = 0

no\_of\_words = len(word\_tokenize(s))

for w in word\_tokenize(s):

if w in tf\_idf\_doc.keys():

sm = sm + tf\_idf\_doc[w]

tf\_idf\_s = float(sm) / float(no\_of\_words)

tf\_idf\_sntnc[ac\_s] = tf\_idf\_s

std\_list.append(tf\_idf\_s)

sd = np.std(std\_list)

for i in range(len(nor\_stp\_lmt)):

cased\_s = stp\_lmt\_cased[i]

ne\_list = get\_continuous\_chunks(cased\_s)

ac\_s = sntncs[i]

e = float(len(ne\_list)) / float(len(word\_tokenize(nor\_stp\_lmt[i]))

op = any(char.isdigit() for char in s)

d = 0

if op:

d = 1

words = word\_tokenize(nor\_stp\_lmt[i])

bag = []

for wd in words:

try:

wd = wd.replace("[", "").replace("]", "").replace("(", "").replace(")", "").replace("{", "").replace("}", "")

r = re.compile(wd + ".\*")

except:

print('efrr1')

newlist = list(filter(r.match, legal\_words))

for item in newlist:

if item in nor\_stp\_lmt[i]:

bag.extend(item.split(" "))

myset = set(bag)

g = float(len(myset)) / float(len(words))

tf\_idf\_sntnc[ac\_s] = tf\_idf\_sntnc[ac\_s] + sd \* (0.2 \* d + 0.3 \* e + 1.5 \* g)

sorted\_x = sorted(tf\_idf\_sntnc.items(), key=operator.itemgetter(1), reverse=True)

file\_nm = os.path.join(output\_path, f)

try:

w\_f = open(file\_nm, "w")

except:

print("write file error")

sumr = ""

for pair in sorted\_x:

sumr = sumr + pair[0] + " "

w\_f.write(sumr)

w\_f.close()

if \_\_name\_\_ == '\_\_main\_\_':

input\_path = sys.argv[1]

output\_path = sys.argv[2]

read\_legal\_dict()

cal\_df()

cal\_total\_doc()

try:

cal\_tf\_Idf() #compute tf-idf, named entities & generates summary

except:

print('tf\_df')

**Code Structure and Functionality:**

**1. Imports and Setup:**

At its core, the "Legal Document Analyzer" relies on a constellation of Python libraries and external resources. These critical components include:

Flask: The primary web framework that underpins the application, facilitating user interaction.

Flask-CORS: An indispensable extension for enabling Cross-Origin Resource Sharing, ensuring the seamless handling of requests from diverse origins.

Beautiful Soup (bs4): A versatile HTML parsing library that excels at extracting data from web pages.

Sumy: A library for text summarization that provides an array of algorithms, including the TextRank summarization method.

Spacy: A natural language processing library that plays a pivotal role in entity recognition.

Transformers: A library from Hugging Face, harnessed to work with pre-trained language models, including RoBERTa.

In addition, the code incorporates several regular expressions for pattern matching, such as "clause," "host\_reg," and "terms\_page."

The sentiment analysis model is loaded from a pre-trained RoBfERTa model using the Hugging Face Transformers library.

**2. Flask Initialization:**

The application's foundation is laid as Flask is initialized. The inclusion of Flask-CORS paves the way for handling cross-origin requests, ensuring the application's responsiveness to input from various sources.

**3. Main Route ("/"):**

This central route acts as the entry point to the application and is designed to handle both GET and POST requests. It provides flexibility in terms of input methods, catering to different user preferences:

text: Allows direct input of text within the request.

link: Permits retrieval of content from a URL.

For POST requests, the code checks the request JSON for 'link' or 'text.' For GET requests, these parameters are sought in the request arguments, making it accessible to a wide array of users.

**4. Processing Data:**

The "process\_data" function takes on the mantle of handling and processing the text data provided. This function encompasses several critical steps:

Data Cleaning: The text is meticulously cleaned and prepared for analysis, ensuring that it is in a format conducive to thorough examination.

Clause Identification: The code employs regular expressions to identify and extract key clauses within the text, a fundamental step in simplifying the legal language.

Text Summarization: The TextRank algorithm from the Sumy library is utilized to provide users with succinct and coherent summaries of these essential sections, enhancing comprehension.

Entity Highlighting: The application extends its capabilities to encompass entity recognition using Spacy. Entities, whether they are organizations or specific terms, are not merely identified but also highlighted within the document, greatly enhancing the user's understanding of the context.

Sentiment Analysis: Beyond individual sentence analysis, the application delves into sentiment analysis. It evaluates sentences within the document for sentiment, categorizing them as positive, negative, or neutral, offering users insights into the emotional tone conveyed in the legal text.

Readability Measurement: The application goes one step further by evaluating the readability and complexity of the entire document, making it accessible to individuals from various backgrounds. An external API is engaged to generate readability and complexity scores, providing a valuable metric for comprehensibility.

**5. Text Processing Functions:**

The code incorporates several functions dedicated to preparing text data for analysis. The "prepare\_for\_regex" function plays a pivotal role in cleaning the text, tokenizing sentences, and readying the data for further analysis.

**6. Text Summarization:**

Text summarization is performed using the TextRank algorithm from the Sumy library. The code selects sentences based on specific criteria and formats them for highlighting, making it accessible and comprehensible to users.

**7. Entity Highlighting:**

One of the standout features of the application is its ability to recognize entities using Spacy. Entities are not only identified but also marked with HTML tags, offering users an enriched understanding of the context and key players within the document.

**8. Sentiment Analysis:**

The code conducts sentiment analysis on the extracted sentences using a pre-trained RoBERTa model. This analysis categorizes sentences as positive, negative, or neutral, adding a layer of insight regarding the emotional tone conveyed in the legal text.

**9. Readability Measurement:**

Readability metrics are calculated for the entire text data using an external API. The application provides both readability and complexity scores, offering valuable insights into how comprehensible and complex the document is.

**10. Web Routes:**

The application extends its reach through two additional web routes:

**/text:** This route renders a user-friendly web interface, allowing individuals to input text for analysis without requiring an understanding of the underlying code.

**/submit\_data:** This route streamlines the data submission process, making the application accessible to a broader audience.

**Conclusion:**

In summation, the "Legal Document Analyzer" is a revolutionary tool that harmoniously combines advanced AI technologies with a user-centric design to demystify the complexities of legal documents. It empowers users by not only simplifying the language but also enriching their understanding of the context, sentiment, and readability of the document.

This application represents a significant leap towards a more user-friendly, efficient, and insightful approach to handling legal texts within an ever-evolving legal landscape. Whether you are a legal professional navigating intricate contracts or an individual striving to understand the implications of a legal document, the "Legal Document Analyzer" is your gateway to unlocking the potential of cognitive computing in the legal sphere.

By enhancing the understanding of complex legal documents and providing a wealth of insights, the "Legal Document Analyzer" marks a paradigm shift in the way we approach legal text analysis. It is poised to play an indispensable role in legal professions and for individuals seeking to make well-informed decisions in the intricate world of legal documentation.