Article Summarizer and Classifier

J COMPONENT PROJECT REPORT

Winter 2019-20

Submitted by

ALIM MANJIYANI (17BCE2107)
JOSYULA SAI SHREESH (17BCE2176)

in partial fulfillment for the award of the degree of

B. Tech

In

Computer Science and Engineering

Vellore-632014, Tamil Nadu, India

School of Computer Science and Engineering

May, 2020

Title

Article Summarizer and Classifier

Abstract

Article classification and summarization have a genuinely backhanded connection as article classification fall into classification issues rather than summarization, where it is treated as an issue of semantics. A significant piece of the summarization procedure is the recognition of the point or subjects that are examined in an irregular document. In light of that, we attempt to find whether article classification can aid administered article summarization. This framework aims to summarize and classify a given new/text using NLP and ML techniques. The classification algorithm uses “Random Forest Classification” technique to classify the given text based on a trained model, which uses 2300 classified texts retrieved from BBC news and divided into 5 categories namely, business, entertainment, politics, sports and tech. Using this methodology we were able to achieve an accuracy of 96% .On the other hand, the summarizer algorithm uses the TF-IDF (Term frequency and Inverse document frequency) technique, which measures the frequency and uniqueness of each word and scores the sentences accordingly. The top 10 sentences based on their scores are shown as the summary for the given article. This summarization technique uses an extractive approach rather than an abstractive approach for article summarization. The whole system follows a single-document, extractive and domain-specific approach to meet the required results.

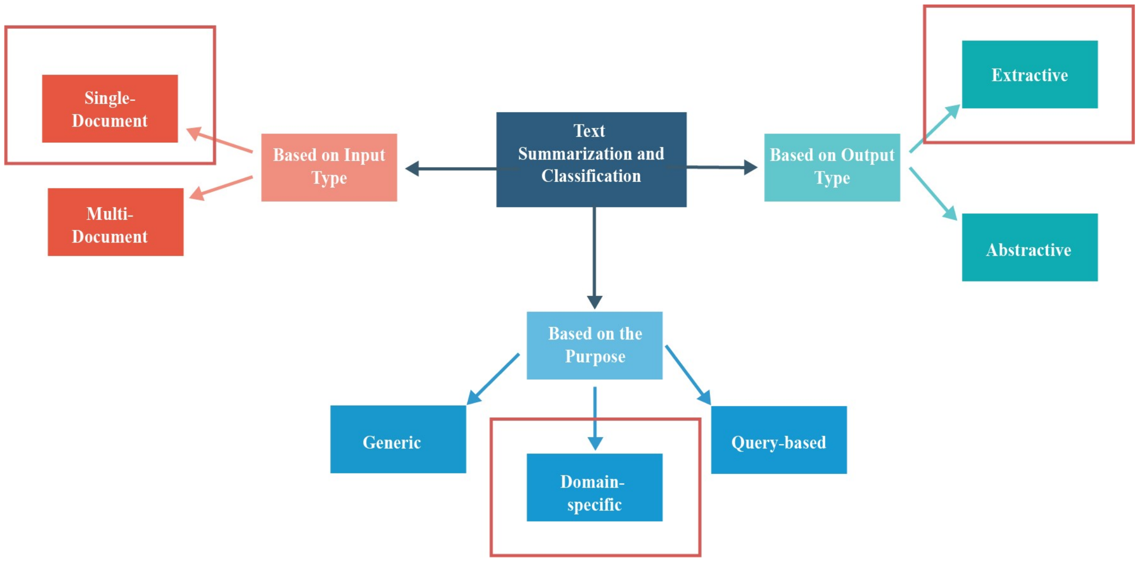
Introduction

With the sensational development of the Internet, individuals are overpowered by the gigantic measure of online data and records. This extending accessibility of records has requested thorough research in the zone of programmed content summarization. As indicated by Radef et al. [11] an outline is characterized as "a book that is created from at least one messages, that passes on significant data in the first messages, and that is no longer than half of the first messages and as a rule, essentially not as much as that".

Programmed content summarization is the assignment of creating a succinct and familiar outline while safeguarding key data substance and generally speaking importance. As of late, various methodologies have been produced for programmed content summarization and applied generally in different areas. For instance, web indexes produce bits as the sneak peaks of the archives . Different models incorporate news sites which produce dense portrayals of news themes ordinarily as features to encourage perusing or information extractive methodologies. Programmed content summarization is testing, since when we as people sum up a bit of text,we for the most part read it totally to build up our comprehension, and thenwrite a synopsis featuring its central matters. Since PCs need human information and language ability, it makes programmed content summarization a very troublesome and non-minor undertaking. There are a number of papers that provide extensive overviews of text summarization techniques and systems[1][7][9].

Thusly, in this paper we center around extractive summarization and domain-specific classification strategies and give a review of probably the most predominant methodologies in this area.

Architecture Diagram



Our project implements an 'Extractive' type Output-type, which summarizes sentences on its score. The sentences with a high score are put together as a summary. Our project falls under Domain-Specific. The 'Domain-specific' category includes Business, sports, Entertainment, Politics, and Tech. Our program compiles one article at a time so it falls under the 'Single document' Input type.

Background Study

[1] Semantic Summarizer (2019).

This paper work talks about the major problems when it comes to summarize and article. It also provides study on single-document and multi-document classification techniques and propose a general evaluation matrix for summarizers

[2] Document relevancy analysis within machine learning systems (2019).

This research work aims to quantify and classify a document relative to a training corpus and select the best match. The training corpus is created with the help of SVM (Support Vector

Machines) Classifier and TF-IDF (Term Frequency – Inverse Document Frequency) vectorization.

[3] An approach based on classifier combination for online handwritten text and non-text classification in Devanagari script (2019).

This article proposes a strategy for dissecting highlights of curved locales and consolidating results of classifiers utilizing Dempster–Shafer Theory (DST) is introduced to order online manually written content and non-content information of any online transcribed record in the most mainstream Indic content—Devanagari. The overall performance is increased by using classifications provided by Support Vector Machines (SVM) and Hidden Markov Model (HMM).

[4] Automatic bengali document categorization based on deep convolution nets(2019).

In this paper, the authors have proposed a framework for Bengali document categorization/classification using deep convolution nets. The given framework consists of classifier models and word embedding (scoring methods). The given framework performs stunningly by giving an accuracy of about 95%.

[5] Media summarizer (2020).

This paper gives a good example of a hybrid summarization technique i.e. include both extractive and abstractive summarization methods. The given prototype consists of an inspector, an array and generator. The inspector identifies the words and the information associated with it, the array is for storing those words, and the generator finally generates the summary, which may include words from the document.

[6] Neural Classifier with Statistic Information of User and Product for Sentiment Analysis (2019).

This paper proposes a novel neutral classifier, which simultaneously fetches and provides statistic information from the input to the neural network. The output of this model yielded an excellent performance in the field of sentiment analysis.

[7] Topic Based Machine Learning Summarizer (2019).

They used domain-based corpus, used suitable corpus for enhancimg the results. Their summarization is expanded their area to context of spotting. The consideration is made that summarization made using extractive type is less effective. They experimented how human can identify that particular category.

[8] Document classification by a hybrid classifier (2019).

In this paper hybrid classifier is used to classify the candidate URL. Naïve Bayes classifier in the hybrid classifier classify the URL of the candidates.they also use sublink classifier which classify the 2nd class of classification which again is classified.

[9] The NLP Techniques for Automatic Multi-article News Summarization Based on Abstract Meaning Representation (2019).

They explained different ways of summarization. About the NLp techniques to automate the multi text news articles etc. the content substance are examined and extricate named elements utilizing Stanford NER apparatus in various perspectives to get dynamic significance of multiple articles.

[10] An Abstractive Summarizer Based on Improved Pointer-Generator Network (2019).

They are proposing text summarization model. Second, the capacity to extricate words from the first content is improved by utilizing the multi-hop attention mechanism, which improves the capacity of the model to process out-of-vocabulary words.

[11] Introduction to the special issue on summarization (2002).

The main goal of a summary is to present the main ideas in a document in less space. If all sentences in a text document were of equal importance, producing a summary would not be very effective, as any reduction in the size of a document would carry a proportional decrease in its informativeness.

Methodology

Modules

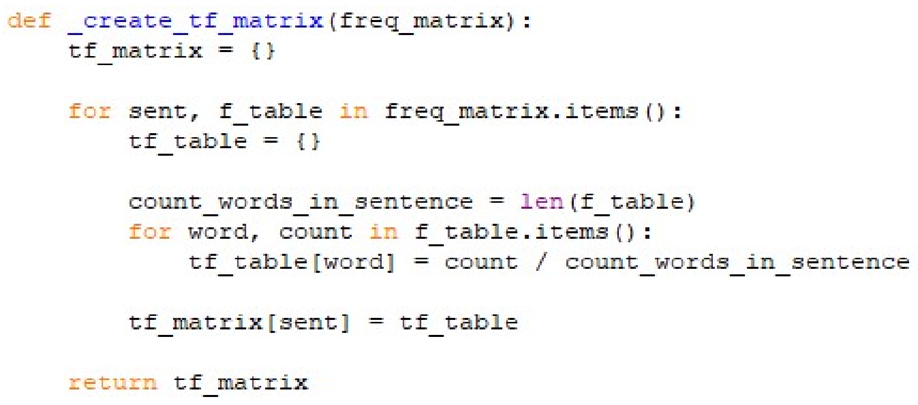
- Summarization
- Classification
- Integration Module

1. Summarization Module:

1.1 Term Frequency and Inverse Document Frequency (TF-IDF) Vectorization Method

1.1.1 Term Frequency:

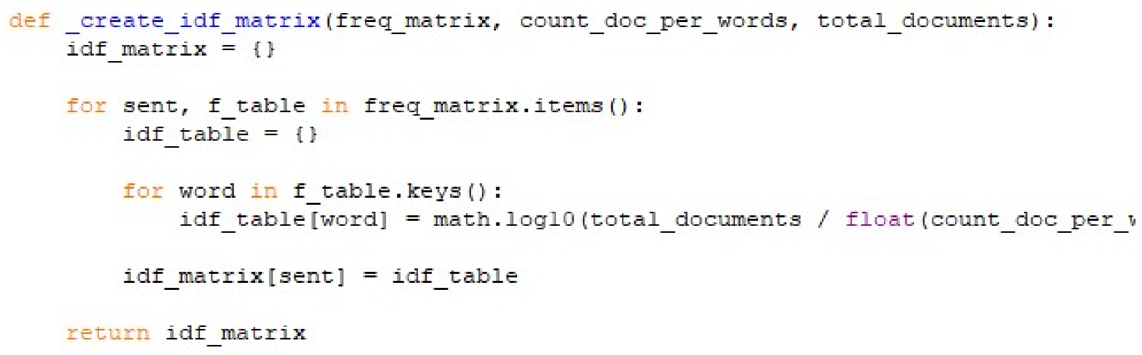
Extracting the words and making a matrix “M1” which contains score for unique words based on their number of occurrence.



```
def _create_tf_matrix(freq_matrix):
    tf_matrix = {}

    for sent, f_table in freq_matrix.items():
        tf_table = {}

        count_words_in_sentence = len(f_table)
        for word, count in f_table.items():
            tf_table[word] = count / count_words_in_sentence

        tf_matrix[sent] = tf_table

    return tf_matrix
```

1.1.2 Inverse Document Frequency:

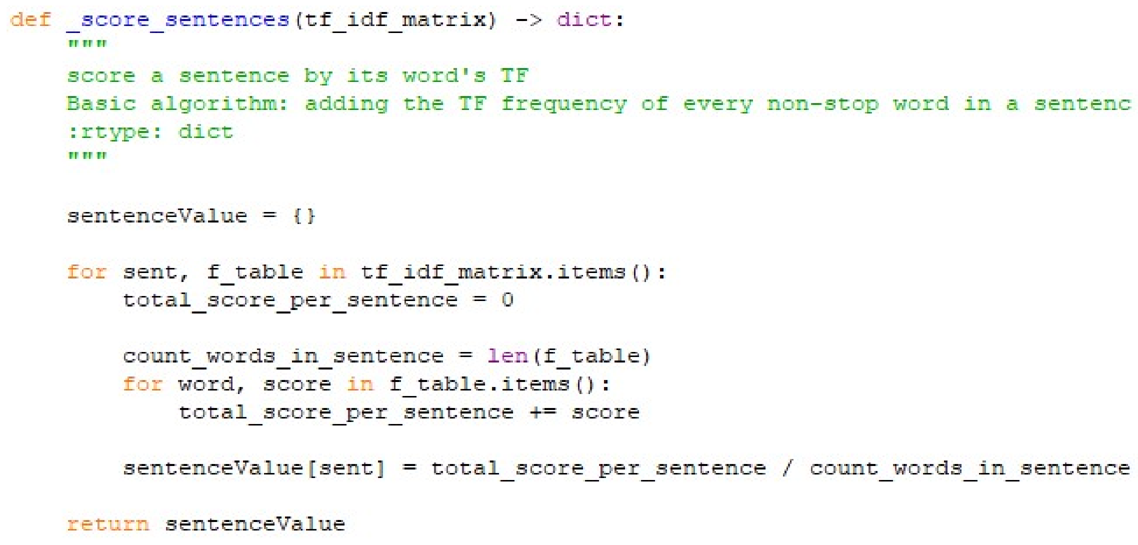
Developing a matrix “M2”which stores the rank of words based on uniqueness of the word in the given document



```
def _create_idf_matrix(freq_matrix, count_doc_per_words, total_documents):
    idf_matrix = {}

    for sent, f_table in freq_matrix.items():
        idf_table = {}

        for word in f_table.keys():
            idf_table[word] = math.log10(total_documents / float(count_doc_per_

        idf_matrix[sent] = idf_table

    return idf_matrix
```

1.2 Calculating Score Method

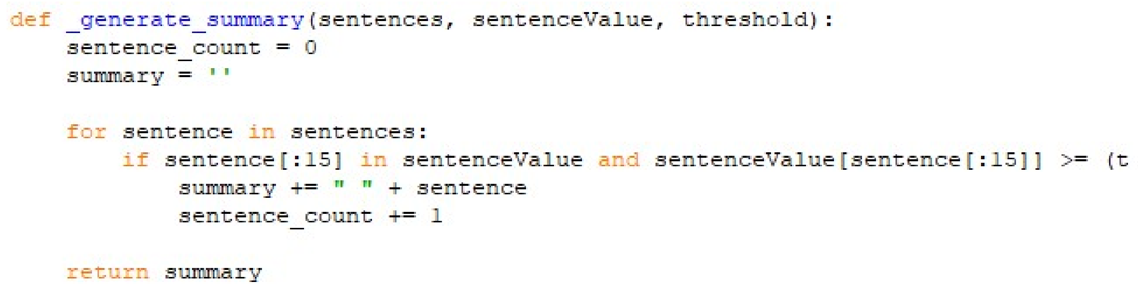
The Score matrix "S" is generated by multiplying the TF-Matrix M1 and IDF-Matrix M2, i.e. S= M1xM2. This matrix (S) gives the final score for each word. Now, the score for each sentence is calculated by summing up the scores of the words present in it.



```python
def _score_sentences(tf_idf_matrix) -> dict:
    """
    score a sentence by its word's TF
    Basic algorithm: adding the TF frequency of every non-stop word in a sentenc
    :rtype: dict
    """

    sentenceValue = {}

    for sent, f_table in tf_idf_matrix.items():
        total_score_per_sentence = 0

        count_words_in_sentence = len(f_table)
        for word, score in f_table.items():
            total_score_per_sentence += score

        sentenceValue[sent] = total_score_per_sentence / count_words_in_sentence

    return sentenceValue
```

1.3 Generating Summary Method

The summary consists of the first 10 sentences with highest scores.



```python
def _generate_summary(sentences, sentenceValue, threshold):
    sentence_count = 0
    summary = ''

    for sentence in sentences:
        if sentence[:15] in sentenceValue and sentenceValue[sentence[:15]] >= (t
            summary += " " + sentence
            sentence_count += 1

    return summary
```

2. Classification Module

2.1 Get_Data Method

This method takes the data from the BBC news dataset, which consists of news categorized into 5 different categories as a text document and convert it into a comma separated value (csv) file, so that it can be accessed by the classifiers.

```
data = {'news': x, 'type': y}
df = pd.DataFrame(data)
print ('writing csv flie ...')
df.to_csv('../dataset.csv', index=False)
```

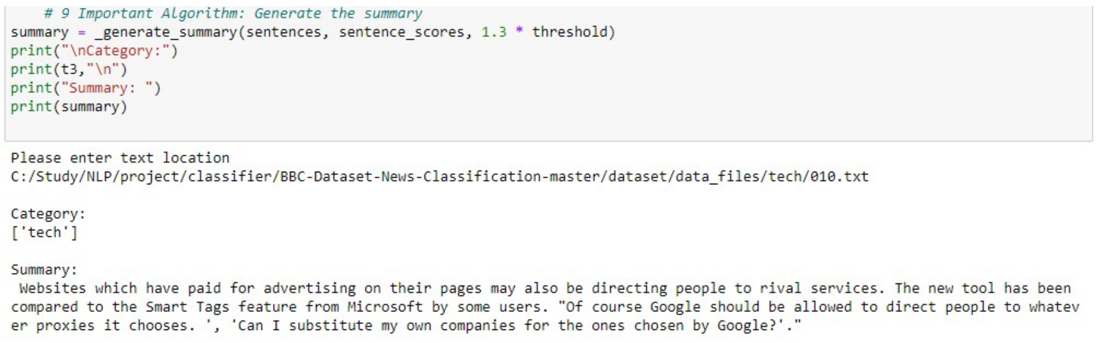
2.2 Creat_Model Method

This method uses the above created .csv file, which acts as a dataset to train the model using Random Forest Classifier which gives the category as an output for the given article

```
model = RandomForestClassifier(n_estimators=300, max_depth=150,n_jobs=1)
model.fit(X_train, y_train)
```

3. Integration Module

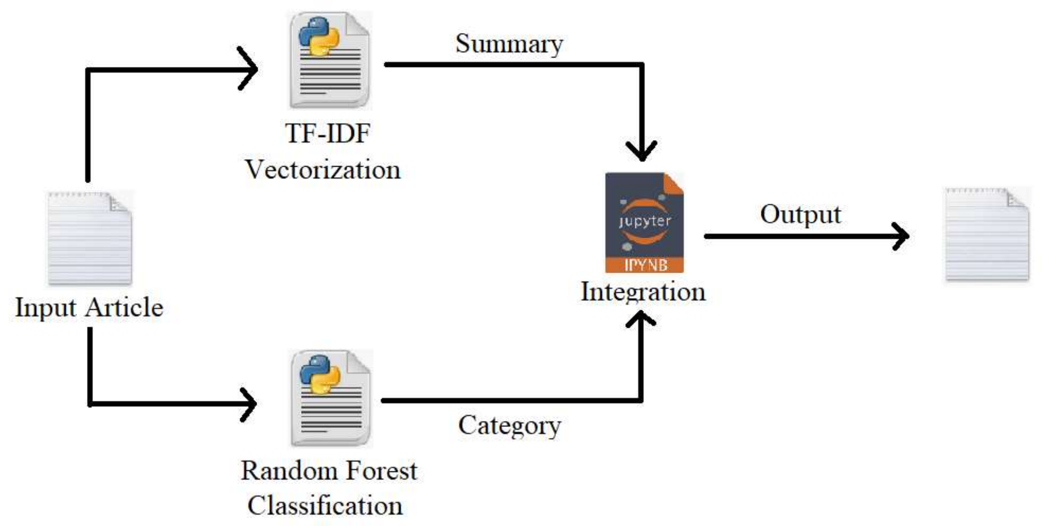
This Module integrates the classification module and summarization module to give the summary and category as a single output.



```
    # 9 Important Algorithm: Generate the summary
summary = _generate_summary(sentences, sentence_scores, 1.3 * threshold)
print("\nCategory:")
print(t3,"\n")
print("Summary: ")
print(summary)
```

```
Please enter text location
C:/Study/NLP/project/classifier/BBC-Dataset-News-Classification-master/dataset/data_files/tech/010.txt

Category:
['tech']

Summary:
 Websites which have paid for advertising on their pages may also be directing people to rival services. The new tool has been
compared to the Smart Tags feature from Microsoft by some users. "Of course Google should be allowed to direct people to whatev
er proxies it chooses. ', 'Can I substitute my own companies for the ones chosen by Google?'."
```

Proposed Model



Input Article – The input text/document/article, which is to be summarized and classified.

TF-IDF Vectorization – Performs TF-IDF algorithm and generated the summary of the given text

Random Forest Classification – Creates a model based on Random Forest Classifier, which trains on the BBC news dataset to categorize the given text

Integration – Receives the Summary and Category of the given text as an input and outputs the same in a presentable manner.

Results and Discussions

Source Code:

1. TF-IDF (Summary Generator):

```
import math

from nltk import sent_tokenize, word_tokenize, PorterStemmer
from nltk.corpus import stopwords

from sys import argv

def _create_frequency_matrix(sentences):

    frequency_matrix = {}

    stopWords = set(stopwords.words("english"))

    ps = PorterStemmer()

    for sent in sentences:

        freq_table = {}

        words = word_tokenize(sent)

        for word in words:

            word = word.lower()

            word = ps.stem(word)

            if word in stopWords:

                continue


            if word in freq_table:

                freq_table[word] += 1

            else:

                freq_table[word] = 1


        frequency_matrix[sent[:15]] = freq_table


    return frequency_matrix


def _create_tf_matrix(freq_matrix):

    tf_matrix = {}


    for sent, f_table in freq_matrix.items():
        tf_table = {}

        count_words_in_sentence = len(f_table)
        for word, count in f_table.items():
            tf_table[word] = count / count_words_in_sentence

        tf_matrix[sent] = tf_table

    return tf_matrix

def _create_documents_per_words(freq_matrix):
    word_per_doc_table = {}

    for sent, f_table in freq_matrix.items():
        for word, count in f_table.items():
            if word in word_per_doc_table:
                word_per_doc_table[word] += 1
            else:
                word_per_doc_table[word] = 1

    return word_per_doc_table

def _create_idf_matrix(freq_matrix, count_doc_per_words, total_documents):
    idf_matrix = {}

    for sent, f_table in freq_matrix.items():
        idf_table = {}
        for word in f_table.keys():
            idf_table[word] = math.log10(total_documents / float(count_doc_per_words[word]))

        idf_matrix[sent] = idf_table

    return idf_matrix

def _create_tf_idf_matrix(tf_matrix, idf_matrix):
    tf_idf_matrix = {}

    for (sent1, f_table1), (sent2, f_table2) in zip(tf_matrix.items(), idf_matrix.items()):

        tf_idf_table = {}

        for (word1, value1), (word2, value2) in zip(f_table1.items(),
                                                    f_table2.items()):  # here, keys are the same in both the table
            tf_idf_table[word1] = float(value1 * value2)

        tf_idf_matrix[sent1] = tf_idf_table

    return tf_idf_matrix

def _score_sentences(tf_idf_matrix) -> dict:
    """
    score a sentence by its word's TF
    Basic algorithm: adding the TF frequency of every non-stop word in a sentence divided by total no
of words in a sentence.
    :rtype: dict
    """
    sentenceValue = {}

    for sent, f_table in tf_idf_matrix.items():
        total_score_per_sentence = 0

        count_words_in_sentence = len(f_table)
        for word, score in f_table.items():
            total_score_per_sentence += score

        sentenceValue[sent] = total_score_per_sentence / count_words_in_sentence

    return sentenceValue

def _find_average_score(sentenceValue) -> int:
    """
    Find the average score from the sentence value dictionary
    :rtype: int
    """
    sumValues = 0
    for entry in sentenceValue:
        sumValues += sentenceValue[entry]

    # Average value of a sentence from original summary_text
    average = (sumValues / len(sentenceValue))

    return average
```

```
def _generate_summary(sentences, sentenceValue, threshold):
    sentence_count = 0
    summary = ''

    for sentence in sentences:
        if sentence[:15] in sentenceValue and sentenceValue[sentence[:15]] >= (threshold):
            summary += " " + sentence
            sentence_count += 1

    return summary


'''
We already have a sentence tokenizer, so we just need
to run the sent_tokenize() method to create the array of sentences.
'''

if __name__ == '__main__':

    f=open(argv[1],"r")
    sentences = sent_tokenize(f.read())



    print(sentences)
    print("\n\n\n")
    # 1 Sentence Tokenize

    total_documents = len(sentences)
```

```
#print(sentences)


# 2 Create the Frequency matrix of the words in each sentence.

freq_matrix = _create_frequency_matrix(sentences)

#print(freq_matrix)



# 3 Calculate TermFrequency and generate a matrix

tf_matrix = _create_tf_matrix(freq_matrix)

#print(tf_matrix)


# 4 creating table for documents per words

count_doc_per_words = _create_documents_per_words(freq_matrix)

#print(count_doc_per_words)




# 5 Calculate IDF and generate a matrix

idf_matrix = _create_idf_matrix(freq_matrix, count_doc_per_words, total_documents)

#print(idf_matrix)


# 6 Calculate TF-IDF and generate a matrix

tf_idf_matrix = _create_tf_idf_matrix(tf_matrix, idf_matrix)

#print(tf_idf_matrix)


# 7 Important Algorithm: score the sentences

sentence_scores = _score_sentences(tf_idf_matrix)

#print(sentence_scores)
```

```
    # 8 Find the threshold
    threshold = _find_average_score(sentence_scores)
    #print(threshold)

    # 9 Important Algorithm: Generate the summary
    summary = _generate_summary(sentences, sentence_scores, 1.3 * threshold)
    print(summary)
```

2. get_data.py

```
import os
import pandas as pd
from sklearn.model_selection import train_test_split

data_folder = "../dataset/data_files"
folders = ["business","entertainment","politics","sport","tech"]

os.chdir(data_folder)

x = []
y = []

for i in folders:
    files = os.listdir(i)
    for text_file in files:
        file_path = i + "/" +text_file
        print ("reading file:", file_path)
        with open(file_path) as f:
            data = f.readlines()
        data = ' '.join(data)
        x.append(data)
        y.append(i)

data = {'news': x, 'type': y}
df = pd.DataFrame(data)
print ('writing csv flie ...')
df.to_csv('../dataset.csv', index=False)
```

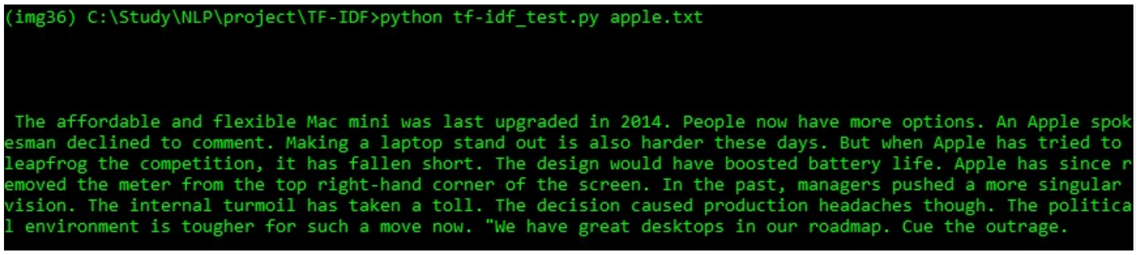
3. Model.py

```
import re
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from textblob import Word
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, cohen_kappa_score, confusion_matrix


def clean_str(string):
    """
    Tokenization/string cleaning for datasets.
    Original taken from
https://github.com/yoonkim/CNN_sentence/blob/master/process_data.py

    """

    string = re.sub(r"\'s", "", string)

    string = re.sub(r"\'ve", "", string)

    string = re.sub(r"n\'t", "", string)

    string = re.sub(r"\'re", "", string)

    string = re.sub(r"\'d", "", string)

    string = re.sub(r"\'ll", "", string)

    string = re.sub(r",", "", string)

    string = re.sub(r"!", " ! ", string)

    string = re.sub(r"\(", "", string)

    string = re.sub(r"\)", "", string)

    string = re.sub(r"\?", "", string)

    string = re.sub(r"'", "", string)

    string = re.sub(r"[^A-Za-z0-9(),!?\'\`]", " ", string)

    string = re.sub(r"[0-9]\w+|[0-9]","", string)

    string = re.sub(r"\s{2,}", " ", string)

    return string.strip().lower()


data = pd.read_csv('../dataset/dataset.csv')

x = data['news'].tolist()

y = data['type'].tolist()


for index,value in enumerate(x):

    print ("processing data:",index)

    x[index] = ' '.join([Word(word).lemmatize() for word in clean_str(value).split()])
```

```
vect = TfidfVectorizer(stop_words='english',min_df=2)
X = vect.fit_transform(x)
Y = np.array(y)

print ("no of features extracted:",X.shape[1])

X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.20, random_state=42)

print ("train size:", X_train.shape)
print ("test size:", X_test.shape)
print(X_test)

model = RandomForestClassifier(n_estimators=300, max_depth=150,n_jobs=1)
model.fit(X_train, y_train)

y_pred = model.predict(X_test)
c_mat = confusion_matrix(y_test,y_pred)
kappa = cohen_kappa_score(y_test,y_pred)
acc = accuracy_score(y_test,y_pred)
print ("Confusion Matrix:\n", c_mat)
print ("\nKappa: ",kappa)
print ("\nAccuracy: ",acc)
```

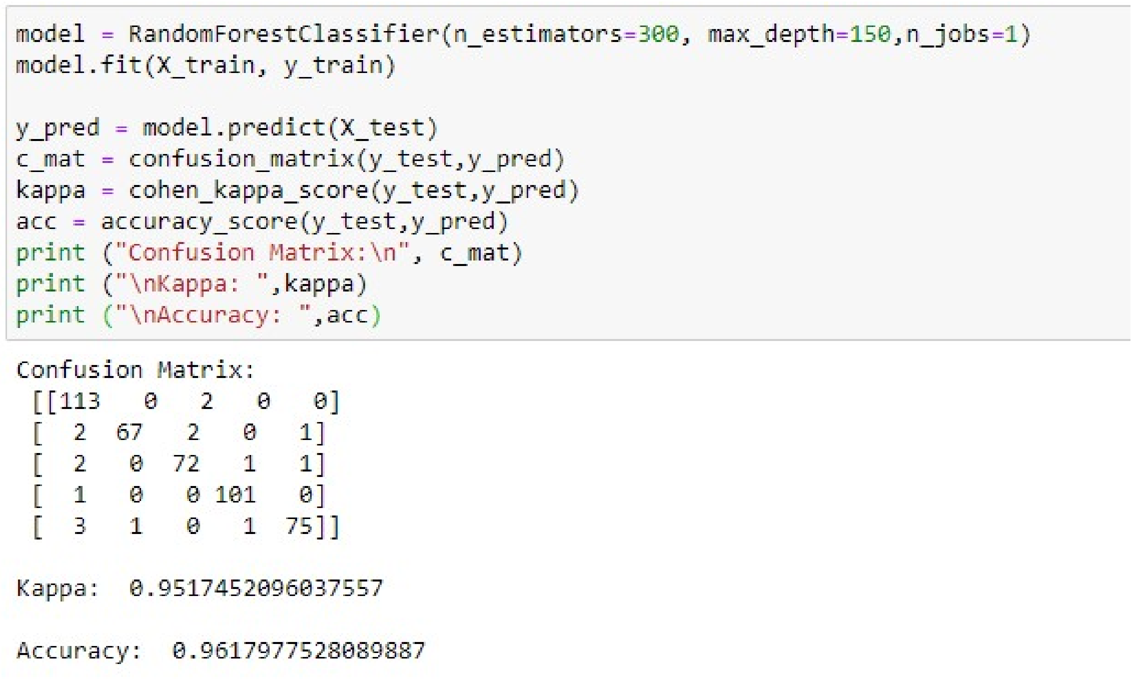
Screenshots

TF-IDF (Module 1)



```
(img36) C:\Study\NLP\project\TF-IDF>python tf-idf_test.py apple.txt



 The affordable and flexible Mac mini was last upgraded in 2014. People now have more options. An Apple spok
esman declined to comment. Making a laptop stand out is also harder these days. But when Apple has tried to
leapfrog the competition, it has fallen short. The design would have boosted battery life. Apple has since r
emoved the meter from the top right-hand corner of the screen. In the past, managers pushed a more singular
vision. The internal turmoil has taken a toll. The decision caused production headaches though. The politica
l environment is tougher for such a move now. "We have great desktops in our roadmap. Cue the outrage.
```

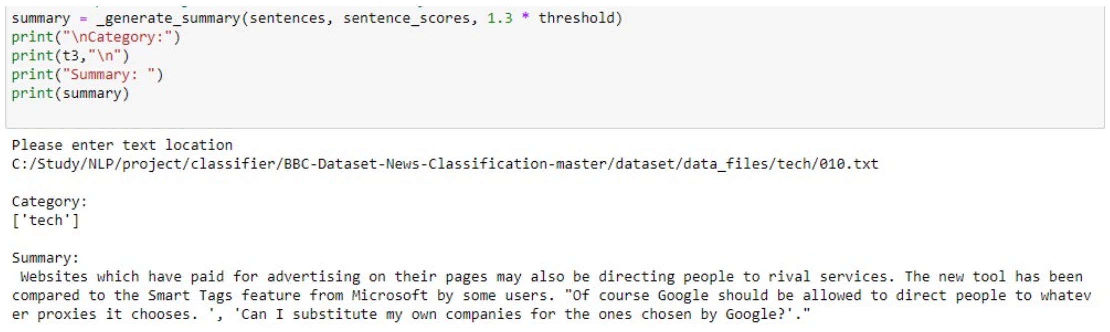
Model.py (Module 2)



```
model = RandomForestClassifier(n_estimators=300, max_depth=150,n_jobs=1)
model.fit(X_train, y_train)

y_pred = model.predict(X_test)
c_mat = confusion_matrix(y_test,y_pred)
kappa = cohen_kappa_score(y_test,y_pred)
acc = accuracy_score(y_test,y_pred)
print ("Confusion Matrix:\n", c_mat)
print ("\nKappa: ",kappa)
print ("\nAccuracy: ",acc)
```

```
Confusion Matrix:
 [[113   0   2   0   0]
 [  2  67   2   0   1]
 [  2   0  72   1   1]
 [  1   0   0 101   0]
 [  3   1   0   1  75]]

Kappa:  0.9517452096037557

Accuracy:  0.9617977528089887
```

Integration.ipynb (Module 3)



```
summary = _generate_summary(sentences, sentence_scores, 1.3 * threshold)
print("\nCategory:")
print(t3,"\n")
print("Summary: ")
print(summary)
```

```
Please enter text location
C:/Study/NLP/project/classifier/BBC-Dataset-News-Classification-master/dataset/data_files/tech/010.txt

Category:
['tech']

Summary:
 Websites which have paid for advertising on their pages may also be directing people to rival services. The new tool has been
compared to the Smart Tags feature from Microsoft by some users. "Of course Google should be allowed to direct people to whatev
er proxies it chooses. ', 'Can I substitute my own companies for the ones chosen by Google?'."
```

Conclusion

The expanding development of the Internet has made a gigantic measure of data accessible. It is hard for people to sum up a lot of content. In this way, there is a massive requirement for programmed summarization devices in this time of data over-burden. In this paper, we underscored different extractive methodologies for single-document summarization. We portrayed the absolute most broadly utilized strategies, for example, frequency-driven methods, graph-based and machine learning techniques Despite the fact that it isn't practical to clarify every different calculation and approaches completely in this paper, we think it gives a decent understanding into ongoing patterns and advances in programmed summarization strategies and depicts the state-of-the-art in this exploration region.

References

[1] Farooq, B. and Hussain, A., 2019. Semantic Summarizer. Journal of Artificial Intelligence Research & Advances, 6(3), pp.1-8.

[2] Feuersänger, C., Wettschereck, D. and Puzicha, J., Open Text Holdings Inc, 2019. Document relevancy analysis within machine learning systems. U.S. Patent 10,324,936.

[3] Ghosh, R., Shanu, S., Ranjan, S. and Kumari, K., 2019. An approach based on classifier combination for online handwritten text and non-text classification in Devanagari script. Sādhanā, 44(8), p.178.

[4] Hossain, M.R. and Hoque, M.M., 2019. Automatic bengali document categorization based on deep convolution nets. In Emerging Research in Computing, Information, Communication and Applications (pp. 513-525). Springer, Singapore.

[5] Jang, E.S., Digital Research Solutions Inc, 2020. Media summarizer. U.S. Patent 10,572,726.

[6] Li, C., Xie, J. and Xing, Y., 2019, October. Neural Classifier with Statistic Information of User and Product for Sentiment Analysis. In CCF International Conference on Natural Language Processing and Chinese Computing (pp. 370-380). Springer, Cham.

[7] Lukyamuzi, A., Ngubiri, J. and Okori, W., 2019, October. Topic Based Machine Learning Summarizer. In 2019 IEEE International Smart Cities Conference (ISC2) (pp. 288-291). IEEE.

[8] Ma, Y. and Cao, X., Fortinet Inc, 2019. Document classification by a hybrid classifier. U.S. Patent 10,313,348.

[9] Nagalavi, D. and Hanumanthappa, M., 2019. The NLP Techniques for Automatic Multi-article News Summarization Based on Abstract Meaning Representation. In Emerging Trends in Expert Applications and Security (pp. 253-260). Springer, Singapore.

[10] Nie, W., Zhang, W., Li, X. and Yu, Y., 2019, June. An Abstractive Summarizer Based on Improved Pointer-Generator Network. In 2019 34rd Youth Academic Annual Conference of Chinese Association of Automation (YAC) (pp. 515-520). IEEE.

[11] Radev, D.R., Hovy, E. and McKeown, K., 2002. Introduction to the special issue on summarization. Computational linguistics, 28(4), pp.399-408.