**RESTAURANT**

**RECOMMENDATION**

**SYSTEM**

**Ayan Chakraborty**

**&**

**Allen Antony**

**INDEX**

1. **Chapter 1- SYSTEM STUDY**
   1. PROBLEM STATEMENT
   2. PROJECT OBJECTIVES
   3. NEED OF THE SYSTEM
   4. SYSTEM REQUIREMENTS
2. **Chapter 2- SYSTEM DESIGN**
   1. INTRODUCTION
   2. DESIGN DIAGRAMS
      1. SYSTEM FLOW CHARTS
   3. WORKFLOW
3. **Chapter 3- CODE OUTPUT**

**BIBLIOGRAPHY**

# 1. SYSTEM STUDY

## 1.1. PROBLEM STATEMENT

## Most of the people find it very difficult to find a restaurant which suits their preferences and needs. So to overcome this problem we have come up with a restaurant recommendation system which helps the user find a suitable restaurant.

The objective of the study is to recommend restaurant according to user preferences.

## 1.2. PROBLEM OBJECTIVES

Conventionally people have to go to each restaurant one after another, to finally find a place to eat and spend time with family and friends after a lot of trial and error. Additionally, there are many uncertainties towards the sources of the price which in fact, most of the price are not retrieved from the actual price in the market. Based on the research, there are many solutions that are available to solve this issue.

## 1.3. NEED FOR THE PROJECT

As customer have come across some problems of the existing system which motivated to develop this application.

Restaurant recommendation model helps user to choose the restaurant based on their favourite cuisines, favourite place to eat, etc. This helps the customer to share pleasant moments with their loved ones, without wasting time by avoiding the hassle of going to each restaurant and eating the food and then regretting it.

## 1.4. SYSTEM REQUIREMENTS

#### HARDWARE AND SOFTWARE REQUIREMENTS

Hardware Requirements (min):

* + - * Intel I3 with 2GHz Processor (For Windows)
      * 2GB Physical memory, 100 GB disk space for application and data.
      * Internet connection for PC. Software Requirements (min):
      * Windows XP or higher version of windows operating system.
      * Python 3.0 or above versions.
      * Jupyter Notebook (Anaconda3)

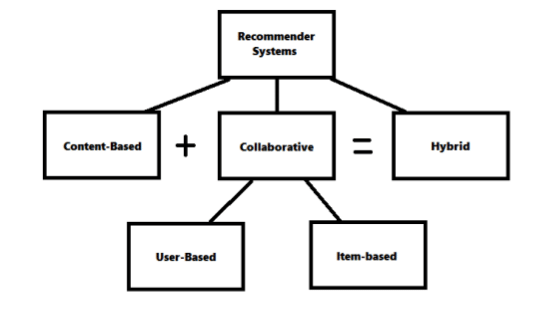
Chrome Web browser version 3.0 or above

**2. SYSTEM DESIGN**

**2.1. INTRODUCTION**

Recommender Systems or Recommendation Systems are simple algorithms that aim to provide the most relevant and accurate items (products, movies, events, articles) to the user (customers, visitors, app users, readers) by filtering useful stuff from a huge pool of information base. Recommendation engines discover data patterns in the data set by learning consumers’ choices and produces the outcomes that co-relates to their needs and interests.

Types of recommender systems:

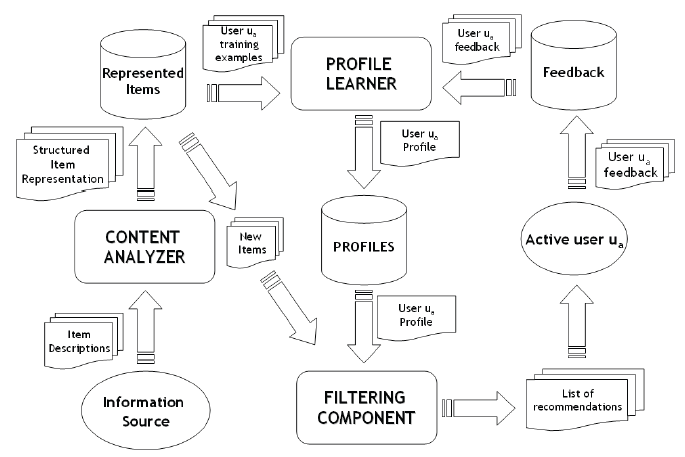


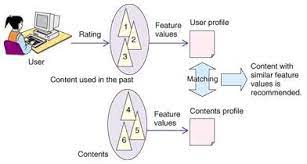
**In this model, Content-based Filtering has been used.**

**Content-based Filtering** is a Machine Learning technique that uses similarities in features to make decisions. This technique is often used in recommender systems, which are algorithms designed to advertise or recommend things to users based on knowledge accumulated about the user.

**2.2. DESIGN DIAGRAMS**

**2.2.1. SYSTEM FLOWCHART**

****

****

**2.3. WORKFLOW**

**Stages of workflow:**

1. **Data Collection**

Collect data related to the topic from a reliable source to work on it.

1. **Data Cleaning**

**a) Removing stopwords**

**Removing stopwords is not a hard and fast rule in NLP. It depends upon the task that we are working on.** For tasks like text classification, where the text is to be classified into different categories, stopwords are removed or excluded from the given text so that more focus can be given to those words which define the meaning of the text.

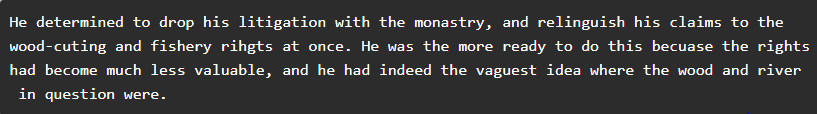
Stopwords are the most common words in any natural language. For the purpose of analyzing text data and building NLP models, these stopwords might not add much value to the meaning of the document. Generally, the most common words used in a text are “the”, “is”, “in”, “for”, “where”, “when”, “to”, “at” etc.

**Advantages**

On removing stopwords, dataset size decreases and the time to train the model also decreases

Removing stopwords can potentially help improve the performance as there are fewer and only meaningful tokens left. Thus, it could increase classification accuracy

**Before:**



**After:**



**b) Normalization**

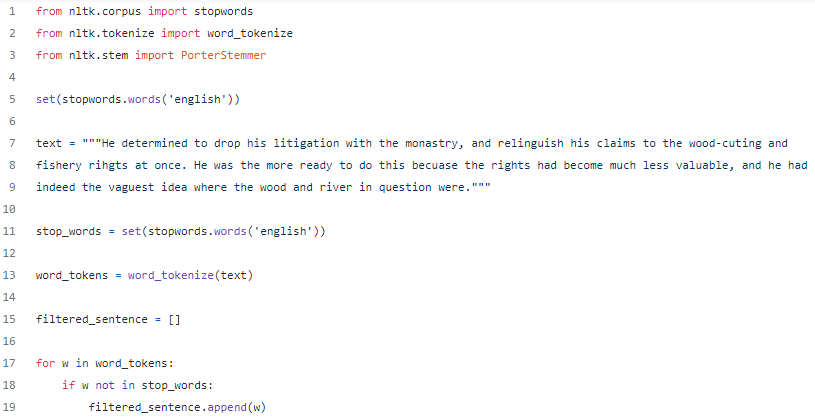
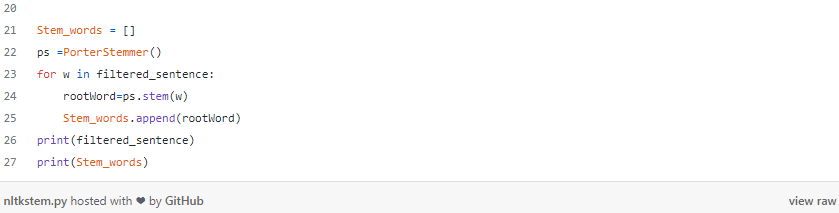
Stemming and Lemmatization is simply normalization of words, which means reducing a word to its root form.

In most natural languages, a root word can have many variants. For example, the word ‘play’ can be used as ‘playing’, ‘played’, ‘plays’, etc.

**Lemmatization -** is an organized & step-by-step procedure of obtaining the root form of the word. It makes use of vocabulary (dictionary importance of words) and morphological analysis (word structure and grammar relations).

**Stemming** **-** is a text normalization technique that cuts off the end or beginning of a word by taking into account a list of common prefixes or suffixes that could be found in that word. It is a rudimentary rule-based process of stripping the suffixes (“ing”, “ly”, “es”, “s” etc) from a word.

Both removal of stopwords & normalization can be performed with the help of NLTK library of python.

****

****

****

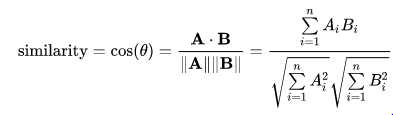
1. **CountVectorizer(Tokenization & Occurrence counting)**

CountVectorizer provided by the scikit-learn library in Python, implements both tokenization and occurrence counting in a single class.

It is used to transform a given text into a vector on the basis of the frequency (count) of each word that occurs in the entire text. It creates a matrix in which each unique word is represented by a column of the matrix, and each text sample from the document is a row in the matrix. The value of each cell is nothing but the count of the word in that particular text sample.

1. **Creating a Cosine Similarity Matrix**

The definition of similarity between two vectors **A** and **B**is the ratio between their dot product and the product of their magnitudes. It is a number bounded between 0 and 1 that tells us how much the two vectors are similar.



sklearn.metrics.pairwise.**cosine\_similarity**(*X*, *Y*, *dense\_output=True*)

Computes cosine similarity between samples in X and Y.

Cosine similarity, or the cosine kernel, computes similarity as the normalized dot product of X and Y:

K(X, Y) = <X, Y> / (||X||\*||Y||)

**Parameters**

**X: *{ndarray, sparse matrix} of shape (n\_samples\_X, n\_features)***

Input data.

**Y: *{ndarray, sparse matrix} of shape (n\_samples\_Y, n\_features), default=None***

Input data. If None, the output will be the pairwise similarities between all samples in X.

**dense\_output: *bool, default=True***

Whether to return dense output even when the input is sparse. If False, the output is sparse if both input arrays are sparse.

**Returns**

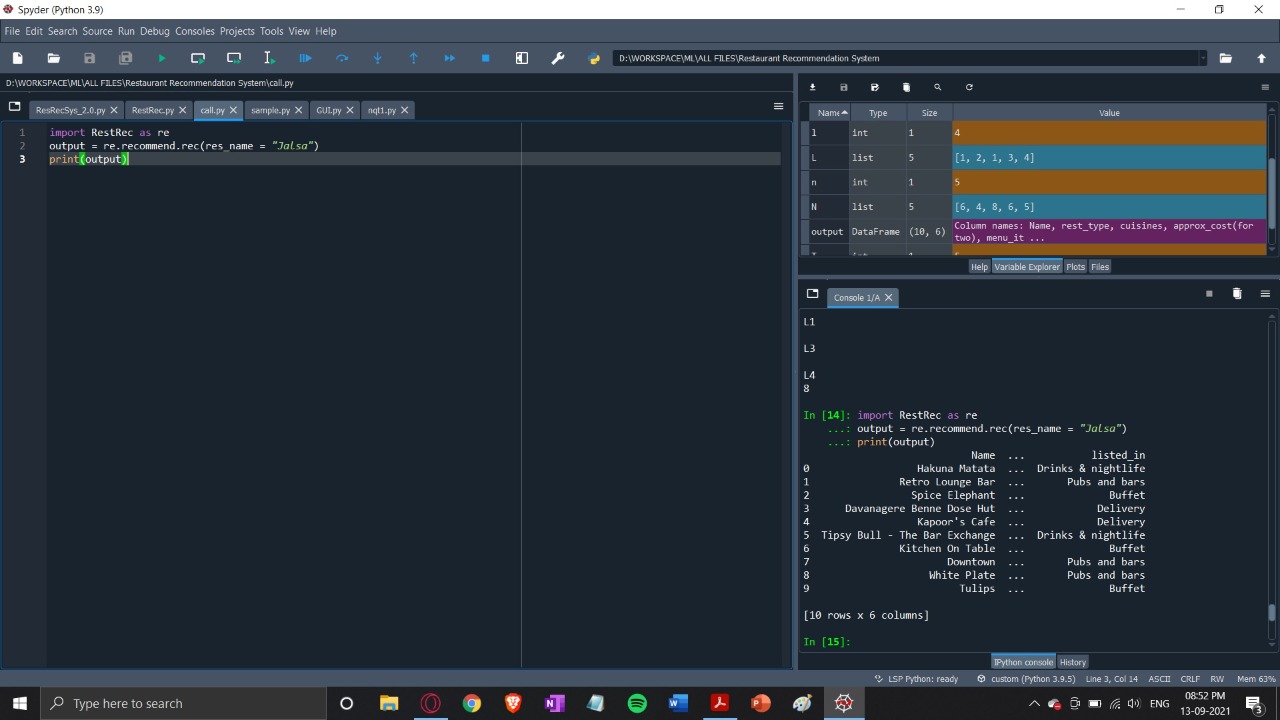
**kernel matrix: *ndarray of shape (n\_samples\_X, n\_samples\_Y)***

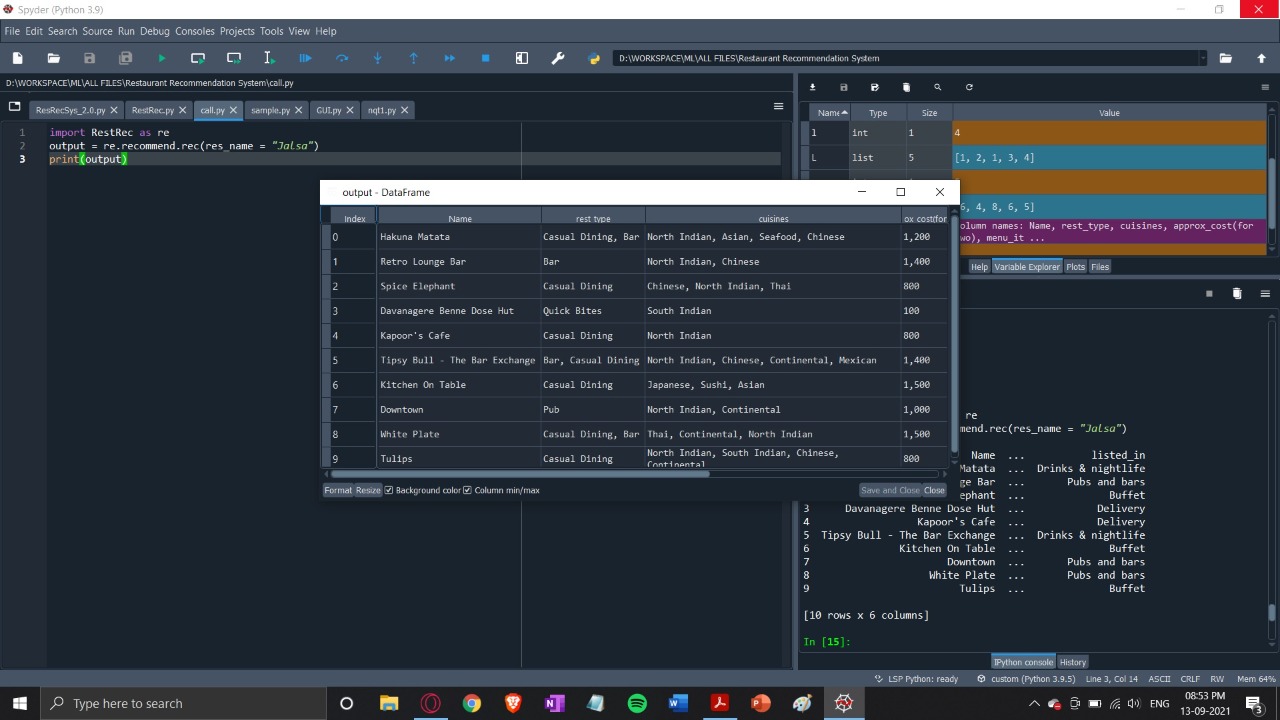
1. **Generating List of Cosine Similarity Scores & Sorting the scores**

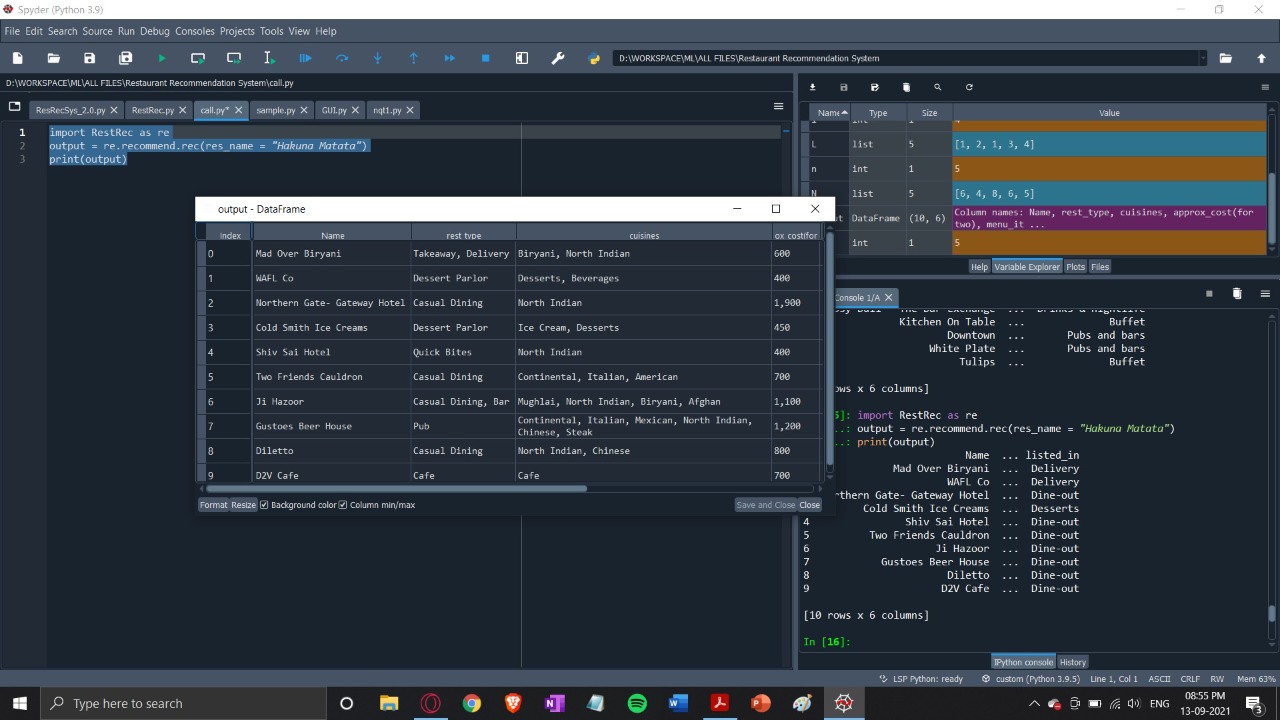
Create the list of cosine\_similarity scores & sort them in descending order.

1. **Display the Recommendations**

**3. CODE OUTPUT**

****

****

****

**BIBLIOGRAPHY**

1. <https://towardsdatascience.com/how-to-build-a-restaurant-recommendation-system-using-latent-factor-collaborative-filtering-ffe08dd57dca>

2. <http://findoutyourfavorite.blogspot.com/2012/04/content-based-filtering.html>

3. <https://www.analyticsvidhya.com/blog/2019/08/how-to-remove-stopwords-text-normalization-nltk-spacy-gensim-python/>

4. <https://scikit-learn.org/stable/modules/feature_extraction.html>

5. <https://www.geeksforgeeks.org/using-countvectorizer-to-extracting-features-from-text/>

6. <https://scikit-learn.org/stable/modules/generated/sklearn.metrics.pairwise.cosine_similarity.html>