Q.2 Implementing search algorithms that provide relevant results is known to be a challenging task because of the complexity of any language. Please complete the below:

Part-1:

Return relevant results considering the following points:

- Different words, sometimes, refer to a common concept.
  Eg.: men, man;
  tree, trees;
- The same word has different meanings given the context.
  Eg: Free parking (no cost for the parking);
  Meat free restaurant (no meat dishes served).

Explain theoretically, (without implementation) how would you use machine learning concepts and techniques to deliver relevant results considering the above points in a search engine context.

Solut'n:

Machine learning(ML) is where a computer teaches itself how to do something, rather than being taught by humans or by following detailed programmings.

Step 1: **Query understanding**

Machine Learning can be used for understanding the search query typed by the user.

Paying more attention to each word in a query coming from user is very important. Ensuring that the whole query (the whole sentence or conversation or meaning) is taken into account, rather than focusing on particular words in our search algorithm.

In query we have to focus on few problems, which are solved by machine learning like:

- Query classification
- Spelling suggestion or spelling correction
- Synonyms / Query expansion (synonyms to expand the query keywords and expand the result set
- Intent disambiguation

Our problem is of **Synonyms/ Query Expansion.**

Step 2: **Candidate Generation**

At this step, given a word, we generate all possible candidates that might be synonyms for the word.

Note: what we mean by "synonyms" usually changes a lot based on domain.

For a search engine, our definition of a synonyms can be much broader and include any alternate words that will help us get better search results for our query.

Eg.: Acronym expansions [CS = "computer science"],
synonyms for proper nouns ["city of gold" = "Dubai"],
or even entire phrase substitutions are good "synonyms" for a search engine. Depending on our exact application, here are some of the sources we can use for synonym generation:

1) **Word embeddings:** we can train word vectors for our corpus and then find synonyms for the current word by using nearest neighbors or by defining some notion of "similarity".

   eg. Word2Vec outputs

2) **Historical user data:** we can also look at historical user behavior and generate synonym candidates from that. A simple way to do this for search engines is to look at word and phrase substitutions by looking for the "query, query, click" pattern. So if we see users searching for [Hotels in Dubai], not clicking on anything, then searching for [rooms in Dubai] and clicking on a result, we can consider "rooms" to be a synonym candidate for "hotels". The simple version of this ignores the context (in this case, the previous word being "Dubai"), but to get more accurate synonyms, we do want to use the previous and next *n* words as context. For word processors, we can similarly look at word and phrase substitutions that our users do.

3) **Lexical synonyms:** These are grammatical synonyms as defined by the rules of the language. Wordnet is a popular source for these among others.

If there are other synonym sources outside of these that are better suited for our application, we should use them too.

We can use either of these techniques in isolation to solve our problem. Eg. a synonym generation algorithm using word2vec vectors alone might be sufficient for us. But by using just one source we will miss out on the strengths that the other sources offer.

Step 3: **Synonym detection**

Now that we have a set of synonym candidates for a given word, we need to find out which ones of those are actually synonyms. This can be solved as a classical **supervised learning** problem.

Given our candidate set, we can generate ground-truth training data either using human judges or past user engagement. As I mentioned above, the definition of synonyms is different for different applications, so if we are using human judges, we will need to come up with clear guidelines on what makes a good synonym for them to use.

Once we have a labeled training set, we can generate various lexical and statistical features for our data and train a supervised ML model of our choice on it.

Note: Any features associated with past user behavior such as word substitution, word frequency perform the best for synonyms.

---------- continue for **Part-2**

Part-2.

A user might engage with the insydo website through different kind of activities.

- How to take into consideration his activities to recommend relevant content and a personalized experience.
- What activities should be tracked?

Solut'n:

Most Online services use artificial intelligence to personalize user experience. User Activities should be tracked to achive a recomandation engin. Typically, there are two types of entities involved in a recommender system (RS), let's call them *users* and *items*. we would like to make recommendations on Items which are the things we would like to recommend to them such as Eat, Drink, Party, Adventure, Explore, , Look Good, Kids, Watch, Travel, and News on 'Insydo'.

A recommendation engine typically processes data through four phases:

1. Collection
2. Storage
3. Analysis
4. Recomandation

1. Collection

A recommendation engine can collect data about users based on their implicit *behavior* or their explicit *input* through searching or "query, query, click" pattern.

Behavioral data is easy to collect because we can keep logs of user activities. Collecting this data is also straightforward because it doesn't require any additional action from the user as they are already using the application. In this approach it's harder to analyze or filtering the interesting logs from the less-interesting users.

Inputted data can be harder to collect because users need to take additional actions, such as writing a review or feedback or experience of the place. Users might not want to provide this data for a variety of reasons. But when it comes to understanding user preferences, such results are quite precise.

2. Storage

The more data we can make available to our algorithms, the better the recommendations will be. This means that any recommendations project can quickly turn into a big data project.

The type of data that we use to create recommendations can help us to decide the type of storage to use. We could choose to use a NoSQL database like MongioDB, or standard SQL database like MySQl, or even some kind of object storage like IBM Cloud object storage.

Each of these options is viable depending on whether we're capturing user input or behavior and on factors such as ease of implementation, the amount of data that the storage can manage, integration with the rest of the environment, and portability.

When saving user ratings or events, a scalable and managed database minimizes the amount of operational tasks needed and helps to focus on the recommendation.

3. Analysis

The first factor to consider in analyzing the data is how quickly we need to present the recommendations to the user.

If we want to present recommendations immediately, such as when the user is viewing a product, we will need a more quick and light in action, type of analysis. Eg., If we want to send the tourist an email that contains recommendations at a later date.

We can follow *Near-real-time* analysis lets us gather data quickly so we can refresh the analytics every few minutes or seconds. A near-real-time system might be good for providing recommendations during the same browsing session.

The platform running the analysis could work directly from a database where the data is saved or on a dump saved periodically in persistent storage.

Another most important aspect of analysis after above disscussed things is ***filtering***. The most common filterning approaches includes:

- **Content-based**: A popular, recommended facts has similar attributes to what the user views or likes.
- **Cluster**: Recommended facts go well together, no matter what other users have done.
- **Collaborative**: Other users, who like the same products the user views or likes, also liked a recommended facts.

Collaborative filtering output is based on the assumption that two different users who liked the same item in the past will probably like the same ones now also is what i feel to be done.

As a Analytic approach, we can represent data about **ratings** or **interactions** as a set of matrices, with iteams and users as dimensions(m rows * n columns). Collaborative filtering tries to predict the missing cells in a matrix for a specific user-product pair.

The process of **Training** the models will be as,

1. **Rank**: The number of unknown factors that lead a user to give a rating. These could include factors such as age, gender, or location etc. The higher the rank, the better the recommendation will be, to some extent. Starting at 5 and increasing by 5 until the recommendation improvement rate slows down, memory and CPU permitting, is a good approach.

2. **Lambda**: A *regularization* parameter to prevent *overfitting*, represented by high *variance*, and low *bias*

Variance :	represents how much the predictions fluctuate at a given point, over multiple runs, compared to the theoretically correct value for that point.
Bias :	represents how far away the generated predictions are from the true value we're trying to predict

The higher the lambda, the lower the overfitting but the greater the bias.

3. **Iteration**: The number of times that the training will run. In this example, we will do 5, 10, and 20 iterations for various combinations of rank and lambda.

And many more aspects to keep in mind in complete analysis process

4. Recomandation

To make the results available to the user quickly and easily, we need to load them into a database that can be queried on demand and provide to user as recomendations.

Practicale implementqtion will be as,
1. Get the data from MySQL / NoSQL / Cloud SQL
2. Convert the DataFrame to fault-tolerant collection of elements that can be operated on in parallel and create the various datasets of it.
3. Train models based on various parameters
4. Calculating top predictions for the user
5. Saving the top predictions
6. Running the solution
7. Monitoring the work of Algorithm

To Track user Activities

We can take help of already availabler User Tracking systems like,
Google Analytics, AWStats, etc.

Or for more granularity we can embed our own set of code in backend which captures and stores every click information of user on website and later can be processed.

Eg. We can use MongoDB to store data(as storing every click) and storing will be done on every click and page load fuctions in backend in Django views.

Google Analytics captures data like:

- **Page Information**
  - URL – the URL of the page the user is viewing
  - Title – the title of the page the user is viewing
- **Browser Information**
  - Browser name – the browser the user is using
  - Viewport or Viewing pane – the size of the browser window
  - Screen resolution – the resolution of the user's screen
  - Java enabled – whether or not the user has Java enabled
  - Flash version – what version of Flash the user is using
- **User Information**
  - Location – this is derived from the IP address where the hit originated. The IP address itself is not available in GA as it is personally identifiable information (PII) which violates the terms of Google Analytics.
  - Language – derived from the language settings of the browser

If we add our set of code to caputer user activities:

- **Features wise tracking**
  - Most viewd feature(user interest) or **Popular Categories.**
  - Repeateadly accesing feature and its sub Categories.
  - Gathering data from social media likes and comments on insydo page and posts.
  - Gettng track of user liking recomendations or not.
    and many more depending on our.