Author: *Varun Rajiv Mantri*

README

Instructions for running

1. Run featureExtracter.py [This code simply extracts distinct features]
2. Follow the prompt and enter the path where you want result files to be stored and hit enter.
3. Also enter path pointing to the directory where the 'user csv(input data)' files are present when prompted.
4. Observe the results in files intermediateOutPut.csv and maxStorage.csv
5. Run inferenceEngine.py [This code classifies users based on their transactional records]
6. Enter path of intermediateOutPut.csv and maxStorage.csv (both need to be in same folder) when prompted.
7. Observe output in Solutions.csv and command line.

Note- Histogram generated shows the distribution of scores of all the users

Requirements-

1. Python
2. Libraries – csv, os, statistics, matplotlib

Part 1: Feature Extraction

- This python code, reads the .csv files in provided directory and extracts essential features which it stores in a file called '*intermediateOutPut.csv'*
- Along with this file one more file called *maxStorage.csv* also gets generated which stores the maximum values for some features.
- **Working**-
    1. Feature extraction starts with search operation. Data in the "*Vendor* column [column No. 4]" is our subject data for search operation.
    2. Program starts with a default list containing '*keyword/key phrase*' which we want to search for in our input file. Every keyword/phrase has been hashed to a specific location in another default list which holds the abstract label (Feature Name) for this keyword/phrase and an index of this Feature Name in yet another list. E.g. – Consider keyword 'taxi' and 'Uber'. Both these keywords are hashed to two distinct locations in a list say LIST_A. At the index in LIST_A where 'taxi' and 'Uber' are hashed, I store another list which holds the '*Feature Name*' and its index in LIST_B. So, LIST_B is our final list which holds all the labels for *extracted features.*
    3. The input string extracted from csv file [column 4] is broken down into separate words and is stored into a list. Combinations of these words (starting with only first word) are hashed using the same hash function and the generated indices are checked for in constant time in the list containing the keywords/key phrases.

4. If the searched word does occur in our default list of keywords/phrases, then a count is incremented for the corresponding feature.
5. Multiplying by the value present in column 5, i.e. the price we get the total amount spent by the individual on a specific item, which later translates into a weight for a distinct feature.

- **Highlight**- The searching algorithm searches for a word/phrase in ***O (1) [constant time].***
- **Rationale –** *Use of Hashing in searching operation of distinct features.*
    1. *Hashing* gives constant time *O (1) access* to the data I am trying to search.
    2. Constant Time access is guaranteed as the keywords/key phrases I am searching are limited and are known a prior. Hence by compromising slightly on space complexity by allocating an array of size 1000 when the elements to be mapped are only 21 I have made sure that collisions do not occur.
    3. As collisions, do not occur, I don't have to use techniques like *open addressing* or *chaining*. Use of any of the above-mentioned techniques would have made worst case time complexity to be *linear time O (n)* [not at all desirable when O (1) is possible]
    4. Thus, by considering the tradeoff between *space complexity* and *time complexity*, I believe I have fine-tuned my algorithm.

Part 2: Inference Engine

- This python code, reads the *maxStorage.csv* file and stores the values extracted into local variables.
- The final output of inference builder is a file called solution which contains *five groups* of users such that all users in any one group are more like each other compared to users from other groups.
- **Working-**
    1. The process of inference buildup starts with *normalization*. The values for the individual features extracted from *intermediateOutPut.csv*, are compared with the maximum possible value for that feature which is extracted from *maxStorage.csv.*
    2. Depending upon which class these features belong to they are weighted differently. e.g. the max value extracted from maxStorage.csv is 5000 and the current value of that feature for one specific user is 4000, then (4000/5000) *25=20. Where '25' is the weightage for that specific feature.
    3. If a feature does have a string value like 'YES/NO' then its contribution to total score is calculated directly. E.g. if feature – A has value 'YES' then its contribution will be 25 and 0 otherwise.
    4. All the weights of features calculated as explained above are summed up to get a total score out of 140. Thus, a python dictionary of a user and his/her individual score is maintained. This score is converted to score between 0-100 by converting it to percentage where 140 is 100%.
    5. Finally, to classify similar users together, standard deviation for all the scores is calculated along with the mean value.
    6. Users are clustered together with following end limits

- ♦ 'Mean +/- 1/2' Sigma (both limits included) ➔ Middle cluster (Group 1)
- ♦ 'Mean +1/2' to 'Mean +1.5' Sigma (upper limit included) ➔ second cluster (Group 2)
- ♦ 'Mean +1.5'Sigma upwards ➔ third cluster (Group 3)
- ♦ 'Mean -1/2' to 'Mean -1.5' Sigma (lower limit included) ➔ fourth cluster (Group 4)
- ♦ 'Mean -1.5' Sigma downwards ➔ Fifth cluster (Group 5)

7. Along with the classification, code also provides facility to take two user-ID's as input and produce score between 0-1 showing compatibility. [1 is 100% compatibility and 0 is 0% compatibility]

- **Highlight-** Use of ***Gaussian distribution*** to categorize similar users together.
- **Rationale-** *Use of Gaussian Distribution to classify users into distinct groups:*
  1. Scores for individual users are generated by taking into consideration distinct features which have been extracted from unbiased behavioral patterns presented in the form of transactional data.
  2. As the transactional data provided for any user has no correlation with the transactional data of any other user, we can infer that scores generated using this information are going to follow random distribution. [This inference has been verified and the results can be seen from the histogram generated during code execution].
  3. As the distribution is random, the curve which best fits this case can be identified to be the "*bell curve* / *Gaussian curve"*.
  4. Using this curve and standard deviation, we can classify users into different categories/groups such that given a user 'A' in one group we know that 'A's compatibility is great with all the other members of his group and not so great with members from other groups.
  5. Thus, in my view classifying data based on Gaussian distribution provides more insight in couple matching problem.
  6. To compare two users directly, difference between their individual scores is taken and the result is divided by 100 to bring the score between 0-1.

Future scope:

- Currently the program considers limited features while calculating score.
- As we go on increasing the number of features, quality of our score improves and we can distinguish two individuals more accurately based on their behavioral traits.
- This means that we will be able to tell predict possible couples more accurately.