**Latent Semantic Analysis using Singular Value Decomposition**

Latent Semantic Analysis (LSA) or Latent Semantic Indexing (LSI) is a natural language processing technique for analyzing the relationship between a set of documents and terms contained in that document and disclosing the underlying relationship between the terms and the documents. LSA uses a matrix called as term-document matrix that describes the occurrences of terms in documents. It is a sparse matrix whose rows correspond to terms and whose columns correspond to documents. The weighting used is term frequency-inverse document frequency (TF-IDF), the element of the matrix is proportional to the number of times the terms appear in each document.

For recommending categories of links to users, the dataset is converted to a matrix similar to the term-document matrix. Each row in the matrix corresponds to a user, and each column in the matrix corresponds to a category. The entries of the matrix are the affinities of the users to the items. The affinities are scaled to fit in the range [-1.0, 1.0]. An affinity of 1.0 means that all votes by the user to the links in that category are up-votes. An affinity of -1.0 means that all the votes for links in that category are down-votes. An affinity of 0.0 means that the user is indifferent to links in that category.

Heuristically, an affinity value that is greater that is 0.5 would imply that the link/category should be recommended to the user, while an affinity value that is less than 0.5 would mean that the link/category does not interest the user much.

**Singular Value Decomposition (SVD)**

Once the term-document matrix has been constructed, Latent Semantic Analysis factorizes the matrix using Singular Value Decomposition (SVD) to analyze the matrix. The reason SVD is useful is that it finds a reduced dimensional representation of the matrix that emphasizes the strongest relationships and throws away the noise. In other words, it makes the best possible reconstruction of the matrix with the least possible information. To do this, it throws out noise, which does not help, and emphasizes strong patterns and trends, which do help. The trick in using SVD is in figuring out how many dimensions or *concepts* to use when approximating the matrix. If too few dimensions are chosen, then important patterns are left out and if too many dimensions are chosen, then the noise caused by random word choices will creep back in.

Singular Value Decomposition (SVD) is a factorization technique that decomposes a real or complex matrix M of dimensions *n* x *m* into matrices U, S and V. Matrix U has dimensions *n* x *d* and contains all the left singular vectors, matrix V has dimensions *m* x *d* and contains the right singular vectors, and matrix S is a diagonal matrix of dimensions *d* x *d* consisting of all the singular values. The value of *d* is chosen to be the minimum of *n* and *m*. The advantage of SVD is that large datasets can be represented in a much compact form and still retain all the similarity information among the rows and the columns in the matrix.

Once the left and singular matrices, U and V, have been computed using SVD, the vectors can be implicitly plotted onto to a *d*-dimensional concept space. Each row in the left singular matrix, U, represents a user, and each row in the right singular matrix, V, represents a category. The users and the category can be implicitly plotted onto the same *d*-dimensional concept space. Clusters of users and categories can be formed so that similar users, similar categories and categories that users are interested in are grouped together.

During the online stage, to make a recommendation of a category to a user, the distance between the user and the category in the *d*-dimensional space is computed using a suitable distance metric. If the distance is less than a particular threshold value, then the category is recommended to the user. Otherwise, it is concluded the category would not be of much interest to the user.

This algorithm is majorly content-based as recommendations are made by only computing the distance between the users and items into account. The other users in the same cluster are not given importance while making recommendations for a user. Since SVD is computationally expensive, the decomposition step, where the model is computed, is performed offline. The model has to be periodically recomputed to accommodate influx of users and categories. This algorithm requires that all the users and items, for which recommendations are to be made, to have gone through the training phase. To compute recommendations for any new user or category, the model has to be re-computed by applying SVD on the new dataset.

**Algorithm**

**Training Phase**

The training phase is the offline stage of the algorithm and consists of computing the model from the dataset. The algorithm employs Singular Value Decomposition (SVD) to decompose the dataset into compact matrices that retain the similarity among users and categories as the original dataset and suppress noise in the original dataset.

1. Create a term-document matrix, M, from the dataset. The dimensions of M are *n* x *m*, where *m* represents the number of users and *n* represents the number of categories.
2. Each entry in the matrix is normalized representation of the number of up-votes over the total number of votes by the user on links belonging to that category. The normalized affinities are in the range [-1.0, 1.0].
3. Apply Singular Value Decomposition (SVD) on the matrix M to factorize M into matrices U, S and V such that:
4. Retain only the left singular matrix, U, and the right singular matrix, V.

**Prediction Generation**

For each pair of target user and target category (ut, at), the following steps are performed:

1. If the user ut or the category at is not among the trained users or categories respectively, then the prediction cannot be made.
2. The distance between the user ut and the category at in the implicit *d*-dimensional concept space is calculated using the Euclidean distance metric. This is done by selecting the vector *x* from the matrix U corresponding to the target user ut, and the vector *y* from the matrix V corresponding to the target category at, and calculating the Euclidean distance between them using the formula:
3. If the distance is less than the threshold distance, then the recommendation is positive. Otherwise, the recommendation is negative.

The threshold distance is chosen by cross-validating the algorithm on the dataset for different threshold values. Parameter optimization techniques such as grid optimization can be used to select the threshold value that produces the best performance.

**Complexity**

The SVD of a matrix M of dimensions *n* x *m* is typically computed by a two-step procedure. In the first step, the matrix is reduced to a bidiagonal matrix. The time-complexity of this step is *O(nm2)*. The second step is to compute the SVD of the bidiagonal matrix. This is performed iteratively, until a certain precision is attained. For *m* iterations, if the precision is considered constant, then the time-complexity of this step is *O(m2)*. The total time-complexity of the training phase of the algorithm is just *O(nm2)*. This step is performed offline periodically; based on how frequently new users and categories are added to the dataset.

The online stage of the algorithm just involved calculating the Euclidean distance between the target user and the category. This operation can be performed in *O(m)* time-complexity for each user-category pair. This makes computing recommendations extremely fast, once the model has been built.

The space complexity of the offline stage is *O(nm)*, since this requires storing the entire matrix to compute the left and right singular matrices using SVD. Once SVD is done, it is enough to store just the left and the right singular matrices, which take up a total space of *O(nd + md)*, where *d* is the number of singular values.In a typical system, the number of users, *n*, will be far greater than the number of categories, *m*. Hence, the space complexity of the online stage can be approximated to *O(nd)*.

**Experiments and Results**

The only parameter that the algorithm requires is the *threshold* Euclidean distance less than which a target (user, category) pair is assigned positive recommendation. Choosing an optimal value for *threshold* is significant to achieve the best performance out of the algorithm.

**Optimizing the parameter *threshold***

Grid optimization is performed over different possible threshold values and the corresponding accuracies for the dataset *attr100* after 10-fold cross-validation are tabulated.

|  |  |
| --- | --- |
| ***threshold*** | **Accuracy in %** |
| 1.04 | 67.45 |
| 1.05 | 73.15 |
| 1.06 | 76.50 |
| 1.07 | 80.80 |
| 1.08 | 77.62 |
| 1.09 | 75.65 |

Optimizing the parameter *threshold*

It can be inferred from the above table that the performance of the algorithm steadily increases with increase in *threshold*, reaches a peak at a *threshold* value of 1.07, and then falls rapidly. The optimal value of *threshold* is thus fixed at 1.07 and is used for all the further experiments.

An advantage with this algorithm is that the training phase is independent of any parameter. Hence, the model can be built using SVD offline, and all further parameter optimizations can be done using the pre-computed left and right singular matrices.

The plot depicting the performance of LSA-SVD on *attr100* dataset for different *threshold* values is shown below:

Parameter *threshold* vs Accuracy

**Performance of LSA-SVD on the datasets**

Experiments were performed on both the *attr100* and *attr200* datasets using the LSA-SVD algorithm with the optimal *threshold* value of 1.07. The following tabulation shows the performance of the algorithm on these two datasets:

|  |  |
| --- | --- |
| **Dataset** | **Accuracy in %** |
| attr100 | 80.80 |
| attr200 | 86.39 |

Performance of LSA-SVD on the datasets

The confusion matrices, along with the precision and recall values for *attr100* and *attr200* datasets, with a *threshold* of 1.07, are presented in the table below:

|  |  |
| --- | --- |
| TP = 15134 | FN = 113 |
| FP = 4182 | TN = 571 |

Precision = 78.34%, Recall = 99.25%

**Confusion Matrix for *attr100* dataset on LSA-SVD**

|  |  |
| --- | --- |
| TP = 14281 | FN = 143 |
| FP = 3964 | TN = 1612 |

Precision = 78.27%, Recall = 99.00%

**Confusion Matrix for *attr200* dataset on LSA-SVD**

It is quite interesting to observe that for both the datasets, even though the recall of the algorithm is of the order of 99%, the precision is only of the order of 78%. The high recall means that the algorithm returned most of the relevant results, or in other words, was correct about most of the positive recommendations. But, the relatively low precision means that the algorithm also returned an insignificant amount of irrelevant results, i.e., the algorithm predicted a large number of target (user, category) pairs as positive recommendations, while they are actually not. This means that the LSA-SVD algorithm produces too many noisy recommendations.

With respect to prediction accuracy, the performance of the LSA-SVD algorithm is both good and scalable. For *attr200* dataset, the accuracy of 86.39% produced by the algorithm is a valid indicator of the scalability of the algorithm. Hence, for datasets that do not change frequently, and for systems that do not frequently encounter new users or categories, the model can be computed offline, so that high-quality recommendations are made during the online stage at a faster rate.