CSE 515 Project Phase II

Zackary Crosley, Tirth Shah, Tithi Gupta, Dhiren Tejwani, Tithi Patel, Riddhi Patel

***Abstract*— Reducing dimensionality of data for analysis and storage is vital in the modern age of big data. Limitations in processing speed force reduction of the dimensionality to ensure timely results can be delivered to user. This paper presents functionality which enables the identification of a variable number of latent semantics and applies these latent semantics for dimensionality reduction of term vectors from a real word data set using SVD, PCA, and LDA algorithms. This requires the loading of the data into a fast but representational data structure, which was performed using the pandas library for python. Finally, we apply these same techniques to perform CP decomposition of a tensor.**

***Keywords***— **dimensionality reduction, vector spaces, similarity, singular value decomposition, principal component analysis, latent dirichlet allocation.**

1. Introduction
2. *Terminology*

CP Decomposition: Canonical Polyadic Decomposition.

KNN: K-Nearest Neighbor. Gets the K closest vectors.

TF: Term Frequency is defined as the number of times are the given “term” appears in a document. For computing the TF, the common words such as ‘the’, ‘is’, ‘are’ should not be counted.

DF: Document Frequency is defined as the number of times the “term” appears in a given specific document. It is important as we can check the relevancy of the document with the help of the query.

TFIDF: Term Frequency- Inverse Document Frequency is defined as the statistical measure used to find the importance of the word in that document. More the word appears, more the importance. It is offset by the frequency of the word in corpus. Models used for the visual descriptors are as follows:

CN: Color Naming Histogram.

HOG: Histogram of Oriented Gradients.

CM: Color Moments.

LBP: Locally Binary Patterns.

CSD: Color Structure Descriptors.

GLRLM: Gray Level Run Length Matrix.

SVD: Singular Value Decomposition. It is the factorization of the data matrix. This data matrix undergoes the eigen decomposition. After decomposition it results into D=USV t where U(object-object matrix) is the left eigen vectors of the DD T matrix. V(feature-feature) is the right factored matrix of the eigen vectors of D T D matrix. S is the core diagonal matrix which consists the values of square roots of eigen values. After decomposition and selecting the top k features, data matrix remains preserved.

PCA: Principal Component Analysis. It is the dimensionality reduction technique which undergoes the factorization of the covariance matrix. This covariance matrix is square and symmetric and undergoes eigen decomposition. After decomposition it results into C=USV T where U is the left eigen vectors of the covariance matrix. V is the right factored eigen vectors of the covariance matrix. S is the core diagonal matrix which consists the values of square roots of eigen values in decreasing order. After decomposition and selecting the top k features, variance remains preserved.

LDA: It is abbreviated as Latent Dirichlet Allocation. It is the dimensionality reduction technique. It is a generative statistical model that allows the set of observations to be explained by unobserved groups. Our Dataset consists of number of documents, each observation consists of observations i.e. words. Each document posits a mixture of large number of topics and each word presence is attributed to one of the document topics.

Cosine Similarity: It is the similarity measure, used to identify the most similar images or locations or users as per the given ID. It performs the dot product between the two vectors and normalizes these vectors by the unit length. It results into the score range [0,1] after normalizing.

1. *Goal Description.*

The goal of this project is to produce a program which demonstrates knowledge and understanding of dimensionality reduction and latent semantics on a real world data set. Using the dataset specified, the deliverable would perform decompositions of various data combinations and present the latent semantics [1]. In specific, this entails as described for the following tasks.

For task 1 we have to implement a program which works on the three different vector spaces. It’s up to user to select either of the user-term vector space or image-term vector space or location-term vector space. After selecting the vector space, user has to input the value ‘k’ as argument during the run time so that the top-k latent semantics for the corresponding term space is found using either of the dimension reduction/latent semantic methods. The latent semantic methods are PCA, LDA, SVD. After finding the latent semantics, we have to express it in terms of the old dimensions with the corresponding weights. These term-weight pairs should be sorted in decreasing order of weights.

For task 2 we have to extend the task 1 in such a way that, after the top-k latent semantics are identified, given the ID (user ID/ Image ID/location ID), system should also find most related 5 user IDs, image IDs, location IDs. Also, we have to list overall matching scores of these IDs in decreasing order.

Task 3 is related to the visual descriptors. For the given visual descriptor model, we have to find the top-k latent semantics using any of the dimension reduction methods (PCA, SVD, LDA). After reducing the latent semantics and expressing it in terms of the given visual descriptors, image ID is passed as input during run time and *n* most related image IDs should be identified. Also using this image ID, *n* most related location IDs should be identified. These image IDs and location IDs are sorted in decreasing order by their overall matching score such that the most related ID has the highest matching score.

Task 4 is also related to the visual descriptors. For the given location ID and the visual descriptor model, we have to identify the top-k latent semantics of that location using either of the dimension reduction methods. We have to list these top-k latent semantics in terms of the old dimensions in order of the weights. After identifying the top-k latent semantics, we have to find the most related *n* location IDs to the given location ID and also list the overall matching score of the corresponding ID and sort them in decreasing order such that the most related ID has the highest overall matching score.

In task 5 the user is asked to give a location ID from the dataset and a value of k.The user also is asked to put his choice for the dimensionality reduction algorithm from the given three algorithms: (PCA, SVD, LDA).Using the ID and the value of the k the task must implement the dimensionality reduction algorithm according to the choice entered by the user. The dimension reduction is to be performed on all the visual descriptor models of the location ID entered. We must identify the top k latent semantics and identify a list of the most 5 related locations considering the obtained latent semantics. The matching score of the top 5 locations with the entered location ID is also to be displayed.

Task 6 asks the user to enter the value of k. We are asked to form a location-location similarity matrix and perform SVD on this similarity matrix. The top- k latent semantics should be extracted and displayed. Each top k latent semantic should be displayed in terms of the location-weight pair in decreasing order of the weights of the location.

In task 7 we have to implement a program with the k entered by the user. A tensor with 3 modes (user-image-location) is to be formed taking into consideration the number of terms shared by them in the devset text file. We have to perform the rank-k CP decomposition on formed tensor and then form k groups of each mode. (k groups of users, k groups of locations, k groups of images). The formed k group of each mode must be non-overlapping i.e for k-groups of the user, each user ID must be in one group only.

1. *Assumptions.*

* For textual descriptors since no model was specified we used *df* for all parts of this program.
* For Singular Value Decomposition we used TruncatedSVD from the scikit-learn library. This is a modified SVD algorithm which truncates the data based on the desired number of components before performing the SVD decomposition. This was assumed to be sufficient for our purposes.
* Since only a *k* value was specified for task 1 in the sample inputs, it was run with *user* term space.

1. Description

This project required several complex features and was thus separated into modules for easy organization and to encourage code reuse. This section will begin by going over the high level modules and then proceed into specific explanations of how specific functionality was implemented. The high level modules that will be discussed the Interface, Loader, Database, Neighbor, and Decompose modules.

The interface module provides the means for a user to interact with the code. It specifies an interface for each task to read in the appropriate inputs, validate the input, and present output (including errors). A ‘help’ command was also added to specify the user interface to the programmer, using python doc strings to produce a modular method of presenting options as code was added. A quit command also allows the user to gracefully end the program. For more specifics on interface, read the section below.

The loader module provides code to read the files from the dataset into intermediate structures before final storage. While memory hungry, the intermediate structures allowed for the separation of final data storage from the loader to prevent changes in one module from affecting the other. Since several data storage schemes were tried before one was selected, this separation was crucial to fast testing and implementation. The loader included several subcomponents. A generic reader class implemented recursive functionality for reading individual files from folders or even directories of folders. Since the dataset was separated into many files, this abstraction became very useful to enabling this functionality for reading multiple different file types. Individual readers inheriting from this for loading textual descriptors (desctxt folder in dataset), visual descriptors (descvis folder in dataset), and location data (devset\_topics.xml and poiNameCorrespondences.txt file in dataset). Finally, the loader component oversees the calling of these individual readers and passing their intermediate representation to the database.

The database module was the component that underwent the most evolution. Finding a representation that was sufficiently fast for our purposes resulted in eventually selecting Pandas dataframes. Pandas had its own limitations, however, as many of the tasks required subdivisions of the data, particularly the visual descriptors. Producing these subdivisions are time consuming when splitting an individual data frame. To improve efficiency, the database stores all the data in separate pandas dataframes and merges them as needed. The locations are stored as a single dataframe with the location information, though it is largely used for correlating names and titles with the location id. The textual descriptors are stored as a dictionary where each description key (user, poi, photo) points to the corresponding dataframe. This has the additional benefit of limiting the number of features compared to combining all the text descriptors into one dataframe, since only the features for this key are present. The textual descriptors are never combined or modified, so this dataframe is static throughout execution. The visual descriptors are stored similarly with each location id and visual model pair point to their dataframe. Each dataframe has all the photos for that location with the feature set for that model. For this project we needed to be able to get all photos for all locations under a model and all models for a location. This was fortunately very fast to run by simply appending dataframes, an operation that appeared faster during testing. To construct a dataframe for all locations in a model, each dataframe for a new location was just appended to the end. Since pandas stores these rows with pointers, this turns out to be a near O(1) operation, making it incredibly fast. To construct a dataframe for a location with all models, each model was merged on index with their features added as new columns. While the directions specified that we should select how to perform dimensionality reduction to prevent operations on this resulting table from being too time expensive, pandas is efficient enough that such optimizations were unnecessary. A single recursive method call handled the construction of the desired visual descriptor dataframe based on the location and model (or lack thereof) provided.

The neighbor module implement a knn search on dataframes using efficiencies provided by the numpy and pandas library. Using an arbitrary distance measure (provided in another file called distance.py) that would calculate the distance from a vector to reach row of a table using matrix operations, the knn would sort these outputs and filter it to the first k items. For additional speedups, the knn was implemented with multiprocessing capabilities. In multiprocessing, the table was split into a variable number of parts and each process calculated the distance from the vector to all items in the table subsection, storing them in an ordered heap. Once all processes completed the heaps would then be merged resulting in an ordered list. While this speedup seemed useful, the datasets from this project were too small for it to be necessary, and it is currently not being used.

Finally, the decompose module performs the decompositions use for dimensionality reduction. This library uses the Truncated SVD, PCA, and LDA decompositions provided by the scikit learn library. These libraries produce the object matrix in terms of the latent semantics, as well as provide the latent semantic basis vectors used for decomposition. Since these methods operate on numpy arrays and return arrays without the useful pandas methods that the rest of the code uses, a decorator was produced for each of these methods to change them back into a pandas dataframe with the appropriate indexes and columns.

*Task 1.*

From these modules, implementing many of the tasks became trivial. For task 1 we implemented a program which works on the three different vector spaces (user, photo, location) and find the top *k* latent semantics. This uses data from the user, image, and location textual description data files. As mentioned in assumptions, we use the df model data. User input indicating the term space, the number of latent semantics, and method are used to determine the dataframe to pull from the database, returning an *n* x *m* database representing all the n objects in relation to the m features. If the user selected user data, for instance, this would return the *m* feature values for each of the n user ids. This *n* x *m* dataframe is passed to the decomposition specified by the user (PCA, LDA, SVD) with the number of latent semantics to identify. The decomposition returns the objects in terms of the latent semantics and the *k* latent semantics. Since we only care about the latent semantics, we throw out the object-latent semantics matrix and print out the latent semantics. The latent semantics are sorted in descending value for each of the original features and presented to the user. For ease of use, these latent semantics are also copied to a csv file for more extensive viewing.

*Task 2.*

For task 2, we want to find the object values in terms of the latent semantics found in task 1 and use these to find the nearest neighbors of this vector. Once again the user specifies the vector space (user, photo, or location), the number of latent semantics to identify, and the decomposition method. In addition to these, the user provides the number of neighbors to find and the id to find the neighbors of. The initial process of decomposition is repeated as stated above but we throw out the latent semantics and keep the object matrix. We then pass this matrix to the knn neighbor functionality with a value k provided by the user which ranks vectors by their distance and we present the top k on screen. For this section, the knn uses L3 distance metric.

*Task 3.*

There are 30 locations in the dataset, each with 10 visual descriptor models. From the 10 visual descriptor models, user has to choose 1 model. The program retrieves the particular data frame based on the user input as specified above. From the retrieved data frame, we have to pass the value for *k*, where *k* is the number of top latent semantics. These top *k* latent semantics are identified using one the three decomposition methods (SVD, PCA, LDA). The method for the dimensionality reduction is up to the user and is passed as the argument during the runtime. The top *k* latent semantics are expressed in terms of the old dimensions with the corresponding weights.

The program then uses the user provided image id to identify the most related image IDs using the cosine similarity measure. To perform this measure, the dot product is calculated between given image ID and other image IDs and then we normalize it to calculate the score within the range of [0,1]. More the score, more similar are the image IDs. User has to input the value of *i* in order to identify most related *i* images. The overall matching score will be sorted in decreasing order such that most related images is at the top.

For part-3 for task 3, we need to identify the most related location IDs for given image ID. The image ID corresponds to the location for at which the photo was taken. The most related locations can be identified after comparing each image ID from each location ID for given visual descriptor model. Now, for the given image ID, we performed dot product between given image ID and each image ID of that location. Then we sum up the score for that location and normalize it by the number of image ID present in that location. We also perform cosine similarity of given image ID with image IDs from other locations and for each location we sum up all their scores and normalize it by that location’s number of image IDs. The final score will be in the range of [0,1]. More the score, more similar are the locations. The user is expected to pass the image ID and value of the most related location IDs to be identified. The result will have most related *i* location IDs with their overall matching score sorted in decreasing order, where the most related location is at the top.

*Task 4.*

In the dataset there are 30 locations each with 10 visual descriptor models. For the given location ID, it is up to user to choose visual descriptor model and pass it as an argument during run time. The database is called and the corresponding data frame for that location and particular visual descriptor model is retrieved. Now, using either of PCA, SVD, LDA, one of the method is chosen by user to identify the top *k* latent semantics. Each of the top *k* latent semantics is represented in terms of all the visual descriptors along with their respective weights.

Now, for a given location ID, we need to identify the *i* most related location IDs for the same visual descriptor model. On reducing the dimensions of each of the locations to k latent semantics, all the image IDs are then mapped onto the given location ID’s latent semantics. In order to find the relativity between the two location IDs, we have used cosine similarity as the similarity measure. The dot product is performed between given location ID and each of the location IDs. The result is normalized and score is obtained in range of [0,1]. The overall matching score is listed in decreasing order and top *i* location IDs are displayed as output.

*Task 5*

For a location ID entered by the user, a table is formed with all the image IDs of that location ID as the first column and the 10 visual descriptors appended to the table column-wise. The SVD, PCA and LDA is performed on this table using ‘sklearn’. The output is saved in “image-semantic-task5.csv”. The output saved in this file is (image Ids of the entered location ID × top-k latent semantics) matrix where the image IDs are the columns and latent semantics are the rows. We get the contribution of each image ID to each k latent semantic. Apply the same selected dimensionality reduction algorithm on all the location IDs for finding the most similar 5 locations with the entered ID. With the reduced features of each location we form the similarity matrix between the given location id and with each other location. The matrix has cosine similarities between image ids of given location and image ids of other location. Hence the value is always between 1 and 0. (giving 1 when two image IDs are same). For the matching score we use a similarity function on this similarity matrix that gives us a scalar matching score of that particular location ID from the entered location ID. The similarity function is as follows:

* The maximum value of the entire matrix is found, and its corresponding entire row and column are removed. We add this value to a variable ‘sum’.
* We keep on finding the maximum values of the matrix and adding this value to ‘sum’ after removing the corresponding row and column. This step is performed recursively for the minimum (number of rows, number of columns) times.
* The final sum gained is then divided by the average number of total number of rows and columns in the matrix.

On the basis of the matching score obtained we sort the locations on the decreasing order and get the top 5 most related locations.

*Task 6*

We need to create a location-location similarity matrix. This similarity between the locations is performed on visual descriptors taking all the models for all the locations. We construct a similarity matrix between two locations. The rows are image ids of one location and columns are image ids of another. The values are the cosine similarity between the two image ids. This is will be 1 if both image ids are same, or else a value between 0-1. Once we have a similarity matrix between two locations, we find its similarity score (similar to task 5). The similarity score function takes this similarity matrix between two locations as its input. The logic is as follows:

* The maximum value of the entire matrix is found, and its corresponding entire row and column are removed. We add this value to a variable ‘sum’.
* We keep on finding the maximum values of the matrix and adding this value to ‘sum’ after removing the corresponding row and column. This step is performed recursively for the minimum (number of rows, number of columns) times.
* The final sum gained is then divided by the average number of total number of rows and columns in the matrix.

This way, if both the locations are similar, we will get the score as 1, or else the score is a value between 0 and 1.

We use the above function to find similarity score between each location pairs. Once we have similarity between each location pair, we create a location- location matrix. The output is saved to “Task6\_SimilarityMatrix.csv”. We perform SVD on the location- location similarity matrix using the sklearn library. We represent the relation between the latent semantics and the locations in the “task6transposetable.csv” file. We then present each latent semantic in form of location name- weight pairs in the decreasing order of the weights.

*Task 7*

To form a tensor of user, image and location, we first find the number of similar terms in all the three text files. By parsing each of the text file, we find the number of similar terms (shared) between the user, image and location. Each entry in the Tensor is the number of shared terms. Using this we form a 3D array. A 3-mode tensor is formed using “tensorly”. The rank-k CP decomposition of this formed tensor is done using “parafac”. Parafac takes the tensor and the rank k as its input. By decomposition we get the three factor matrices: (user × latent semantics), (image × latent semantics) and (location × latent semantics). Using the individual factor matrices, we find k groups of users, k groups of images and k groups of the locations which are non-overlapping. To form the k groups, we use k-means algorithm which partitions the entire set of the users in k clusters. The output gives us the group number and the users/images /locations belonging to that group number. The result is saved in a text file. (task7UserGroups.txt, task7ImageGroups.txt, task7LocationGroups.txt).

1. *Interface Description*

*Task 1.*

Command at Prompt: task1 <term space> <k> <method>

Term Space: String text description type (user, photo, poi)

K - Int number of latent semantics to return.

Method - Decomposition to run (PCA, SVD, LDA)

*Task 2.*

Command at Prompt: task2 <term space> <k> <method> <j> <id>

Term Space: String text description type (user, photo, poi)

K - Int number of latent semantics to return.

Method - Decomposition to run (PCA, SVD, LDA)

J - Int number of nearest terms to find.

ID - String id to find nearest neighbors for (id type depends on term space selected).

*Task 3.*

Command at Prompt: task3 <visual model> <k> <method> <id>

Visual Model - String visual description model to use (CM, CM3x3, CN, CN3x3, CSD, GLRLM, GLRLM3x3, HOG, LBP, LBP3x3)

K - Int number of latent semantics to return.

Method - Decomposition to run (PCA, SVD, LDA)

ID - Int image id to find nearest neighbors for.

*Task 4.*

Command at Prompt: task4 <location id> <visual model> <k> <method>

Location ID - Int location id to find semantics for.

Visual Model - String visual description model to use (CM, CM3x3, CN, CN3x3, CSD, GLRLM, GLRLM3x3, HOG, LBP, LBP3x3)

K - Int number of latent semantics to return.

Method - Decomposition to run (PCA, SVD, LDA)

*Task 5.*

Command at Prompt: task5 <location id> <k> <method>

Location ID - Int location id to find semantics for.

K - Int number of latent semantics to return.

Method - Decomposition to run (PCA, SVD, LDA)

*Task 6.*

Command at Prompt: task6 <k>

K - Int number of latent semantics to identify.

*Task 7.*

Command at Prompt: task7 <k>

K - Int number of non-overlapping groups to create.

1. System and Installation Requirements

The program is designed to run in Python 3.6.0

and above. Once Python 3.6.0 or above is installed, use the Pip installer to add the following libraries:

* lxml 4.2.5
* Numpy 1.15.2
* Pandas 0.23.4
* Scikit-Learn 0.20.0
* Tensorly 0.4.2

To execute the code, execute the main.py file from command line with python 3.6 and the libraries installed. (IE terminal:$ python3.6 main.py)

1. Related Work

Not Applicable.

1. Conclusions

While it is difficult to validate the accuracy of our output due to the data being far too large to compute by hand, it appears that the output of our code is correct. The dimensions of all output are as expected with reasonable values based on the visible patterns in the data. In our teams testing we found that many of the decompositions produced are similar to or the same knn results as our results from phase one, operating on non-decomposed data. This suggests successful identification of latent semantics, it would be unlikely for random or semi-random vectors to produce that same result. Time wise task one to task four execute within seconds, while task five and six execute within fifteen to twenty; a reasonable delay for the amount of data being processed. Task seven takes many minutes, however we were told to expect it to be slow.

Our team was able to successfully identify the latent semantics of arbitrary term data and print them in descending order as required for task 1. Task two was able to apply these latent semantics to object representation and use those to compare object vectors by distance measures. The third task then found the latent semantics of photos using a specific visual model to identify latent semantic and calculate the closest images to a provided one by latent semantics. Similarly task four finds the latent semantics for locations under a visual model and then finds the nearest locations by these latent semantics. Task five builds on this and performs the same task with all visual models. Task six found the latent semantics of a location-location similarity matrix. Finally task seven we were able to successfully create and then decompose the tensor, albeit with considerable execution time.

1. Bibliography
2. *B. Ionescu, A. Popescu, M. Lupu, A.L. Ginsca, H. Muller, Retrieving Diverse Social Images at MediaEval 2014: Challenge, Dataset and Evaluation, MediaEval Benchmarking Initiative for Multimedia Evaluation, vol. 1263, CEUR-WS.org, ISSN: 1613-0073, October 16-17, Barcelona, Spain, 2014.*
3. Appendix

Zackary Crosley: Database structure and Loader, Nearest Neighbor, Interface, Task 1 and Task 2 Implementation, Report Compilation.

Tirth Shah: Task 3 and Task 4 Implementation. Task 1 through Task 4 write up.

Tithi Gupta: Task 3 and Task 4 Implementation. Task 1 through Task 4 write up.

Dhiren Tejwani: Task 5 to Task 7 Implementation. Task 5 through Task 7 write up.

Tithi Patel: Task 5 to Task 7 Implementation. Task 5 through Task 7 write up.

Riddhi Patel: Task 5 to Task 7 Implementation. Task 5 through Task 7 write up.