CSE 515 Multimedia and Web Databases Project Phase 2

Amey Athale[*], Vineeth Chitteti[†], Shreesh Nayak[‡], Lokesh Sharma[§], Ankit Kumar Rai[¶], Richa Nagda[‖]
ASU School of Computing, Informatics, and Decision Systems Engineering
Arizona State University
Tempe, AZ, UnitedStates
[*]aathale@asu.edu
[†]vchittet@asu.edu
[‡]sdnayak1@asu.edu
[§]lsharma6@asu.edu
[¶]aarai@asu.edu
[‖] rjnagda@asu.edu

Abstract:

In this project phase, we applied concepts of dimensionality reduction on vectors models and multivariate time-series data sets. We used several methods for dimensionality reduction such as PCA, SVD, NMF, etc. as per the requirement. Next, we performed a comparison between two gesture files on the basis of the transformed data and gave a reasonable explanation for the observed results. Here, we have experimented with different cost functions for certain distance metrics like edit distance and dynamic time warping, and chosen the logically sound ones. Finally, as per the project's requirements, we executed programs to cluster the gesture files using techniques like K-Means and spectral clustering. Finally, we compared and contrasted the various results obtained to conclude our experimentation.

Keywords:

PCA, SVD, NMF, LDA, Edit Distance, DTW, K-Means, Spectral Clustering

Introduction:

We would like to begin this report by discussing some key concepts and terms we might see and use in the upcoming sections. We will briefly describe this project phase problem statement and would clearly state the assumptions that we have used in our proposed solution.

Terminology:

The following are some key concepts that we should keep in the back of our minds before moving ahead. These are explained in more detail in the tasks where they are being used.

1. **PCA:** Principal Component analysis is a common dimensionality reduction technique that performs a change of basis on the data, by considering co-variance of attributes.
2. **SVD:** Singular Valued Decomposition is another dimensionality reduction technique that preserves distances and angles in the data rather than variance.
3. **NMF:** Non-Negative Matrix Factorization can be used as a dimensionality reduction technique where we can incorporate additional domain information and constraints.
4. **LDA:** Latent Dirichlet Allocation is another factorization method that uses a generative model by working on distributions that represent the real world.
5. **Edit Distance:** It is a way of quantifying dissimilarities between two sequences by counting the minimum number of operations required to transform one sequence into the other. The operations that are generally performed are Insert, Delete and Replace.
6. **DTW:** Dynamic Time Warping is a time series algorithm that is mainly used for the comparison of two temporal sequences by trying to align them until an optimal match is found. It helps in minimizing the effect of shift and stretch by doing elastic transformations.
7. **K Means:** is an unsupervised machine learning algorithm that identifies k number of centroids, and allocates every data point to the nearest cluster, while keeping the centroids as small as possible. It serves as a way to group data points into k clusters based on how similar they are to each other. K Means iteratively tries to minimize the distance of each point from its centroid and maximize the distance between every cluster centroid.
8. **Spectral clustering:** Laplicaian based spectral clustering is a method to cluster irregular shaped and sized data.

Goal Description:

Task 0a: Gesture word dictionary

The goal of this task is to parse the raw gesture data given to us and generate words based on user input. Along with this, we are required to calculate certain statistical measures to which would be useful in future tasks.

Given a set of user inputs for a window length(w), a shift length(s) and resolution(r), for each gesture file and for each component(or dimension) of that gesture file, corresponding to every sensor, the following sub-problems are needed to be solved:

- Average of the amplitudes of the time series.
- Standard Deviation of the amplitudes of the time series.
- Normalize all the values in every gesture file to lie between -1 and 1.
- Quantize the values into 2r levels.
- Form words of length w by moving a window with shift s.
- For each of the above-created words, calculate average quantized amplitude.

All of the above information will be stored in files with names *f.wrd.*

Task 0b: Calculation of TF, TF-IDF vectors

In this task, corresponding to every gesture file *f.csv*, we will create 2 gesture vectors based on TF values and TF-IDF values. All of these values will be written to files called *tf_vectors_f.txt* and *tfidf_vectors_f.txt* in the same directory.

Task 1: Top k latent semantic/topics

This task will ask the user for an input which will be the vector model to be used and the method for dimensionality reduction to be used. Here, we will calculate the contribution of every word to the *k* features.

Task 2: 10 most similar gestures

In this task, as per the user's input of a gesture file(inside directory *dir*), we will return 10 most similar gestures based on vectors models(TF and TF-IDF) using standard dot product similarity metric or metrics like DTW and Edit Distance as per the user's choice.

Task 3: Latent Gesture Discovery

In this task, we are required to create a gesture-gesture similarity matrix and perform dimensionality reduction techniques like SVD and NMF on this similarity matrix, and finally, return the top p principle components or latent semantics.

Task 4: Latent Gesture Clustering and Analysis

In this task, we are required to group or cluster the above created gesture-gesture similarity matrix and perform different clustering techniques on gestures on the basis of either their degree of membership to the latent semantics or using standard iterative methods like K-Means and Spectral clustering method.

Assumptions:

- All the query files provided as an input during any task should be present in the data directory before running any commands.
- During the entire project phase, whenever we mention a word, we always mean it represents: <Component, Sensor_ID, winQ>. All the .wrd files are stored in the same format.
- In our calculations for TF and TF-IDF, we have considered the entire gesture file as our document, but have differentiated each word according to the component and the sensor it appears in.
- All code executions will be performed using the Python3 interpreter.
- The output of Task 4 is dependent on Task 3, hence, the user option given in both places is the same.
- Task 3 and Task 4 are given the same value of *p*.

- Results are obtained using default values **(w = 3, r = 3, s = 3, p = 4, k = 4, random starts = 100, Number of Iteration = 200)**
- The reported results are shown using the **TF model** for **Tasks 1 and 2** and the **TF-IDF model** for **Tasks 3 and 4**.
- The **similarity measure** used for computing similarity between gesture objects in Tasks 2 and 3 for user options **PCA, SVD, and NMF** is **cosine similarity**.
- In the **KMeans algorithm**, the distance measure used for computing the distance of each gesture object from the cluster centroids is **Euclidean Distance.**
- We are creating 2 different representations of .wrd files, one of which will be used for quick and easy lookup.

Proposed Solution:

For better understanding, the figure below shows the directory structure of the testing data given to us.

./data
    ./data/W
        ↳1.csv
        ↳2.csv
    ./data/X
        ↳1.csv
        ↳2.csv
    ./data/Y
        ↳1.csv
        ↳2.csv
    ./data/Z
        ↳1.csv
        ↳2.csv

Figure 1: Test data hierarchy

Task 0a:

Here, we compute several statistical values and symbolic representations on the above-given data which are needed for future tasks. Now, corresponding to each gesture file, for every component or dimension, for every sensor, we first calculate the average amplitude and standard deviation of the values, represented by terms ***avgi,j*** and ***stdi,j*** respectively. This is done using a standard numpy function as shown in the code block below:

```
sensor_dict['avg'] = np.average(df.iloc[i,:])
sensor_dict['stdev'] = np.std(df.iloc[i,:])
```

Code 1: Calculates average amplitude and standard deviation

Next, we normalize the values between -1 and 1 by using the standard normalization formula as shown in the equation below:

$$x = 2 * \frac{x - min(x)}{max(x) - min(x)} - 1$$

Equation 1: Normalizes values in *x* between -1 and 1

For us, this x will be the values in each sensor. The code snippet used for this part is shown below where df is a Pandas DataFrame containing a component of a gesture file.

```
df.iloc[i,:] = 2 * ((df.iloc[i,:] - min(df.iloc[i,:])) /
      (max(df.iloc[i,:]) - min(df.iloc[i,:]))) - 1
```

Code 2: Normalization using pandas

Next, for quantizing the data into *r* Gaussian bands, we used equation 2 to calculate the length of each band. This was implemented via *scipy.integrate* library function provided by python.

$$length_i = 2 * \frac{\int_{(i-r-1)/r}^{(i-r)/r} Gaussian_{(\mu = 0.0,\ \sigma = 0.25)}(x)\ \delta x}{\int_{-1}^{1} Gaussian_{(\mu = 0.0,\ \sigma = 0.25)}(x)\ \delta x}$$

Equation 2: Calculating band lengths

Using these above created non-uniform lengths, we calculated the non-uniform ranges corresponding to these lengths, and assigned each data point to its respective band. For the purpose of this report, we have used the input *r = 3* and we obtained the following ranges:

[(-1, -0.992402100078635),
(-0.992402100078635, -0.8176293512220891),
(-0.8176293512220891, 0),
(0, 0.8176293512220891),
(0.8176293512220891, 0.9924021000786352),
(0.9924021000786352, 1)]

Figure 2: Ranges obtained for *r = 3*

Each band here is represented by the center of the range. In addition to this quantization, we represent each band or bucket by a symbol, which is zero-indexed. For example, for a given input r = 3, we will get 6 buckets, which are symbolically represented by 0,1,2,3,4,5.

Now, on this data, we move a window of length *w* over each sensor(row) by shifting it by *s* units at a time and create a list of tuples, where each tuple contains symbols which semantically represent a word of length w. These tuples are called the symbolic quantized window descriptor $\mathbf{winQ_{i,j,h}}$ for window h, of sensor j, belonging to component cl of gesture file i. Along with this word, for each word, we compute an average quantized amplitude $\mathbf{avgQ_{i,j,h}}$, which is basically

the average of the centers of the bands corresponding to each value. The above-generated values are unique to each sensor, and component for a gesture file, hence, we stored all this information in the form of dictionaries for easy and faster lookup. The figure below shows the hierarchy of the *f.wrds* file which contains this newly calculated data.


```
{
        'W':
                {'0':
                        {'avg': 0.0781152,
                        'stdev': 0.012884994249513657,
                        'words': [[[3, 3, 3], -0.4088146756110446],
                                [[3, 3, 3], -0.4088146756110446],
                                ...
                                [[1, 1, 2], -0.9658059419096657],
                                [[2, 2, 2], -0.9050157256503621]]},
                 '1':
                        {'avg': 0.015976135,
                        'stdev': 0.00394933743788436,
                        'words': [[[1, 2, 2], -0.935410833780014],
                                [[2, 3, 3], -0.5742150256241504],
                                [[3, 3, 3], -0.4088146756110446],
                                [[3, 3, 4], -0.13627155853701486], ...
```


Figure: File structure of every .wrds file

As seen above, each file f.wrds will have 4 keys, viz. W, X, Y, Z. For each dimension, we have 20 keys, one for each sensor. Here, each sensor stores the information we calculated above like the averages, standard deviations, and the symbolic representation of words.

Along with this, we stored the same information in *f.wrd* files with <Component, Sensor_ID, word> format, as requested in the project document, and kept these files in a subdirectory called *data*. The structure looks like:


```
{
        'W':
                {'0':
                        {'avg': 0.0781152,
                        'stdev': 0.012884994249513657,
                        'words': [[“W”, 0, [3, 3, 3], -0.4088146756110446],
                                [“W”, 0, [3, 3, 3], -0.4088146756110446],
                                ...
                                [“W”, 0, [1, 1, 2], -0.9658059419096657],
                                [“W”, 0, [2, 2, 2], -0.9050157256503621]]},
                 '1':
                        {'avg': 0.015976135,
```

'stdev': 0.00394933743788436,
'words': [[“W”, 1, [1, 2, 2], -0.935410833780014],
[“W”, 1, [2, 3, 3], -0.5742150256241504],
[“W”, 1, [3, 3, 3], -0.4088146756110446],
[“W”, 1, [3, 3, 4], -0.13627155853701486], ...

Figure: File structure of every .wrds file

Task 0b:

In this task, each word is associated with a TF and TF-IDF value. The word generated in task 0a is used here for the calculation of TF and TF-IDF value. Since our vector space consists of <*component Name, sensor ID, winQ*> as the dimensions the TF and TF-IDF are calculated accordingly. Here each .wrds file contains words belonging to a gesture.

TF-Vector Calculations:

For the calculation of TF value corresponding to a word in a .wrds we count the number of occurrences of a particular word(<*component Name, sensor ID, winQ*>) in that .wrds file and divide it with the total number of words in that .wrds file. TF value defines the description power of an element inside our object. Here the object is the gesture file containing words.

The working of the code is as follows, we initially count the total number of words in the .wrds file, then we loop through each component(W, X, Y, Z) within which we loop through the sensors and get the count of unique words inside the sensor.

TF(w,f) = (Number of times word w appears in gesture file f) / (Total number of words in gesture file)

This count is divided by the total number of words for normalization as we cannot determine a word occurring the same number of times in a longer file to be of same importance as it occurring in a shorter length file. The resultant TF-Value for a word is then stored in tf_vectors_f_i.txt where f_i is the filename of the .wrds file and the structure is the same as the .wrds file. There is no rounding of numbers done it takes the maximum decimal places that python allows.The format of the file is as follows -

tf_vectors_f_1.txt

```
{
        'W':
                {'0': [
                        ((3, 3, 3), 0.0006578947368421052),
                        ((3, 3, 3), 0.0006578947368421052),
                                ...
                        ((1, 1, 2), 0.0006578947368421052),
                        ((2, 2, 2), 0.0006578947368421052)]},
                 '1': [
                        ((1, 2, 2), 0.0006578947368421052),
```

((2, 3, 3), 0.0006578947368421052),
...
((3, 3, 3), 0.0006578947368421052),
((3, 3, 4), 0.0006578947368421052)]}...,

Figure: File structure of every tf_vectors_f_1.txt file in the form of <component, sensor_id, window> with its TF-Value

TF-IDF Vector Calculations:

For the calculation of the TF-IDF value for a word, we multiply the TF-value with the IDF value of the word. The TF value calculated in the above step is used along with the IDF value and the resultant score for the word is stored in tfidf_vectors_f_i.txt

For the calculation of IDF value for a word we count the number of different .wrds files the particular word(<*component Name, sensor ID, winQ*>) occurs and divide it by the total number of the .wrds files in the directory. IDF value describes the discrimination power of a word within a list of objects i.e different gesture files. Hence lower the value of occurrence higher is the discrimination power of the word. The IDF is calculated using the formula -

IDF(w) = log(Total number of gesture files / Count of files containing w)

TF-IDF(w,f) = TF(w,f) * IDF(w)
Here w is the word denoted by <*component Name, sensor ID, winQ*>
f are the different gesture files

The IDF value of the word denotes how rare or common a word is among all the files. A word occurring in all the files will have a lower IDF value than a word occurring in only a single .wrds file among all the .wrds files.

For the calculation of IDF value, we loop through each component(W, X, Y, Z) within which we loop through the sensors, and for each word, we find the word occurrence(count) across files for that particular component_name(W, X, Y, Z) and sensor_id i.e a word will have the same IDF value across all gesture files for that component_name and sensor_id. Both the TF and IDF value is then multiplied to get the TF-IDF value of the word. The value is then stored in tfidf_vectors_f_i.txt where f_i is the filename of the .wrds file and the structure is the same as that of the .wrds file. The format of the tfidf_vectors_f_i.txt file is as follows -

tfidf_vectors_f_1.txt

```
{
        'W':
                {'0': [
                        ((3, 3, 3), 0.0005759662745749341),
                        ((3, 3, 3), 0.0006578947368421052),
                                ...
```

((1, 1, 2), 0.0005789747368421052),
((2, 2, 2), 0.0005789467368421052)]},
'1': [
((1, 2, 2), 0.0005785947368421052),
((2, 3, 3), 0.0005738947368421052),
...
((3, 3, 3), 0.0005759662745749341),
((3, 3, 4), 0.0006578947368421052)]}...,

Figure: File structure of every tfidf_vectors_f_1.txt file in the form of <component, sensor_id, window> with its TF-IDF Value

In both the files tf_vectors_f_i.txt and tfidf_vectors_f_i.txt we have stored the word in a nested form. This way has helped us a lot in retrieva of the TF-IDF values as well as symbolic window descriptors thereby reducing the time taken for parsing input in the subsequent tasks. Here tf_vectors_f_i.txt and tfidf_vectors_f_i.txt are stored in the pickle form to preserve the structure of the data and also making reading/retrieval of data easier for the subsequent tasks.

Task 1:

In this task, we identify the top k latent semantics/topics in the form of (word: score) pairs for each latent semantic, with words sorted in decreasing order of their scores. The input to this task are the gesture files in the form of tf and tf-idf vectors created in task 0. These word score pairs are identified based on the method selected by the user which is either PCA, SVD, NMF or LDA. We have made use of python's sklearn packages to compute the top k latent semantics using PCA, SVD, NMF and LDA

```
from sklearn.decomposition import PCA
from sklearn.decomposition import TruncatedSVD
from sklearn.decomposition import NMF
from sklearn.decomposition import LatentDirichletAllocation
```

#1 PCA Principal Component Analysis

In the Principal Component Analysis dimensionality reduction technique we aim to retain the **top k latent semantics** with the **maximum variance** or in other words, discrimination power of that latent topic.

PCA Computation:

***Step 1*:** Input data D is an n x m matrix where n = number of gestures in the dataset and m = number of features/words in the dataset. Each row in this matrix is the TF or TF-IDF vector representation of the gesture, thus the length of each row is equal to the total number of words in the dictionary.

Step 2: Calculate the Covariance matrix C from the input matrix D. The covariance matrix is an MxM symmetric matrix (where m is the number of words in the dictionary) and the entries store the covariances associated with all possible pairs of the initial variables.

Step 3: Calculate the eigenvectors and eigenvalues of the covariance matrix C. This is done by the eigen decomposition of C into U S U^{-1} where U is the matrix of eigenvectors and S is the diagonal matrix with eigenvalues on the diagonal and zero everywhere else. The eigenvalues on the diagonal of S will be associated with the corresponding column in U — that is, the first element of S is λ_1 and the corresponding eigenvector is the first column of U. This holds for all elements in S and their corresponding eigenvectors in U.

Eigen Decomposition of a Symmetric Covariance Matrix:

$$\mathbf{C = U S U^{-1}}$$

Where,

1. C = *m x m* - Symmetric **Covariance matrix** of original m features
2. U = *m x m*' - **Left factor matrix** which describes the m old features in terms of the m' latent features
3. S = *m' x m'* - **Core matrix** which describes the importance of m' latent features
4. U^T = *m' x m* - **Right factor matrix** which describes the m' latent features in terms of the M old features

Step 4: Sort the Eigenvectors in decreasing order of their Eigenvalues. For example, if λ3 is the largest eigenvalue, then take the third column of U and place it in the first column position. The columns of this new sorted matrix are the same as the columns of U in a different order, since these eigenvectors are linearly independent of one another. Each eigenvalue roughly captures the importance of its corresponding eigenvector.

Step 5: Keep only the top or first k eigenvalues and eigenvectors and drop the remaining vectors. The proportion of explained variance is the sum of the eigenvalues of the top k features or latent topics divided by the sum of the eigenvalues of all features. Thus, the resultant U* matrix would have the dimensions m x k.

$$\mathbf{C' = U^* S U^{*-1}}$$

Where,

1. C' = *m x m* - New **Covariance matrix** with some variance lost
2. U* = *m x k* - **Left factor matrix** which describes the m old features in terms of the top k' latent features
3. S' = *k x k* - **Core matrix** which describes the importance of top k latent features
4. U^{*T} = *k x m* - **Right factor matrix** which describes the top k latent features in terms of the m old features

Step 6: Calculate the new features or representation of the gestures in the feature space in terms of these k latent topics by multiplying the input n x m -> D matrix with the resultant m x k -> U* matrix. The output of this task would be an n x k matrix which represents each of the n objects in the dataset in terms of the new k latent topics.

#2 SVD (Singular Value Decomposition):

Given a matrix of size(m x n), Singular Value Decomposition(or SVD) decomposes the matrix into a ***left factor matrix***(describes the n objects in terms of the k latent features), a ***core matrix***(describes the importance of k latent features), a ***right factor matrix***(describes the k latent features in terms of the m old features).

$$A = USV^T$$

Where :

1. *A* is an *m x n* matrix
2. *U* is an *m x n* orthogonal matrix - ***left factor matrix***
3. *S* is an *n x n* diagonal matrix - ***core matrix***
4. *V* is an *n x n* orthogonal matrix. *T* on top represents transpose of the matrix V - ***right factor matrix***

Since multiplication by orthogonal matrices preserves linear independence, the rank of matrix A is the same as the rank of the core matrix S.

With the above equation, we will have the original matrix A with all its noise intact. However, if we only keep the k largest singular values(from the core matrix), then we have an approximation of the original matrix. Here we are taking this approximation because we assume that the small values are the noise, and most significant patterns in the data can be expressed using values associated with the larger singular values.

#3 NMF(Non-negative Matrix Factorization):

Intuition:

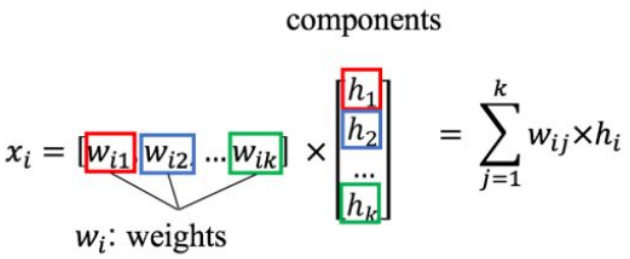
NMP is a matrix factorization method where we constrain the matrices to be nonnegative. NMF decomposes a single matrix into two matrices.

When we decompose the matrix into 2 matrices, there is a chance that they may not represent the original matrix, there by we approximate the values and initialize them with some random values, and iteratively get them as best as we can.

Now suppose that **X** is composed of m rows and n columns and is decomposed into two non-negative matrices **W** and **H**(all the matrices have to be non-negative).

$$X \approx WH$$

Each element in the matrix **X** (***xi***), can be interpreted to be a weighted sum of some components, where each row in H is a component, and each row in ***W*** contains the weights of each component.



We can treat each column(or sometimes row) of the input matrix **X** as the input data point, in that case W becomes a component and H becomes a set of weights.

NMF is an NP-hard problem in general, so initialize the vectors randomly and try to minimize the difference between the initial and final vectors.

$$minimize \|X - WH\|_F^2 \, w.r.t. W, H \, s.t. W, H \geq 0$$

A common optimization algorithm is Alternating Least Squares which can optimize the above minimization problem.

We used ***scikit-learn***'s ***NMF*** implementation to find the decomposition of the input matrix.

#4 LDA(Latent Dirichlet Allocation):

LDA, a popular probabilistic model for collections of discrete data, is a popular method for topic modelling on data such as text corpora. The common terminology in LDA consists of word, document, topic and corpus.

- A ***word*** is the basic unit of discrete data, defined to be an item from a vocabulary.
- A ***document*** is a sequence of N words
- A ***corpus*** is a collection of *M* documents
- A ***topic*** can be seen as a latent semantic which is made up of ***words***

LDA is a generative probabilistic model of a corpus. The basic idea is that documents are represented as random mixtures over latent topics, where each topic is characterized by a distribution over words. LDA discovers topics into a collection of documents. And LDA tags each document with topics.

So given a document(a text corpus), LDA might classify a few words under a topic, we can label these topics as we deem appropriate. Then we can infer each sentence in the document and it's topic allocation(***sentence1 - 10% topic 1, 10% topic 2, 80% - topic 3***). And we can

also derive words that are composed in a given topic(***topic 1 - 30% word1, 20% word 2, 50% word 3***).

LDA Steps:

1. Give an initial estimate of how many topics can be present.
2. LDA will assign every word to a temporary topic.
3. LDA will check and update the word-topic assignment based on parameters such as:
    a. How prevalent is that word across topics?
    b. How prevalent are topics in the document?

The process of checking the word-topic assignment is repeated for each word in every document, cycling through the entire collection of documents multiple times. This iterative updating is the key feature of LDA that generates a final solution with coherent topics.

The input to LDA is documents and the output is topics and their contribution in explaining a document. For example, if the number of topics is 3 and we get the output as <0.2, 0.3, 0.5> from LDA for a document, this can be interpreted as topic 1 being able to explain the document 20%, followed by topic 2 being able to explain the document 30% and topic 3 50%. Thus, we see that topic 3 has the most contribution in explaining the document.

An additional point to note is that the output from LDA is a PDF and hence we use **KL Divergence** to measure similarity.

Output of Task 1:

The output is an ***k x m*** matrix where ***k*** = number of latent semantics and ***m*** = Total number of words(<component, sensor, window>) in the original feature dictionary which essentially describes the k latent features in terms of the m original features.

It has the following format:

Latent Feature 1	<w11, s11>	<w12, s12>	<w13, s13>	…	<w1m, s1m>
Latent Feature 2	<w21, s21>	<w22, s22>	<w23, s23>	…	<w2m, s2m>
Latent Feature 3	<w31, s31>	<w32, s32>	<w33, s33>	…	<w3m, s3m>
…					
…					
Latent Feature K	<wk1, sk1>	<wk2, sk2>	<wk3, sk3>	…	<wkm, skm>

For example: **Results for model = TF, k = 4** and **user option = PCA** are as follows where for each latent topic there is a list of <word, score> pairs sorted in decreasing order of their scores:

"1": [(('W', '10', (5, 5, 5)), 0.2210820110202353),
(('W', '6', (0, 0, 0)), 0.2164060602659026),
(('Y', '6', (1, 1, 1)), 0.19635583750406743),
(('Z', '9', (3, 3, 3)), 0.19589718693135894),
…
…
(('Z', '6', (2, 2, 2)), -0.09607778100283171),
(('Z', '9', (2, 2, 2)), -0.1293830039276335)],

“2”: [(('Y', '6', (3, 3, 3)), 0.1725265635462938),
(('Y', '5', (3, 3, 3)), 0.15058777501215204),
(('X', '9', (3, 3, 3)), 0.1397422000170493),
(('W', '10', (5, 5, 5)), 0.1383447252518842),
…
…
(('Y', '5', (2, 2, 2)), -0.13678509620102736),
(('X', '8', (1, 1, 1)), -0.17674887484690932)],

“3”: [(('X', '11', (3, 3, 3)), 0.1922223980445485),
(('Z', '9', (3, 3, 3)), 0.16136381008402548),
(('W', '8', (2, 2, 2)), 0.14556223456518314),
(('Z', '8', (3, 3, 3)), 0.1248443844608534),
…
…
(('W', '8', (4, 4, 4)), -0.15587437540402277),
(('X', '8', (1, 1, 1)), -0.20288770450363358)]

“4”: [(('Z', '6', (4, 4, 4)), 0.17387926989159772),
(('Z', '10', (2, 2, 2)), 0.13595147570224062),
(('W', '7', (2, 2, 2)), 0.13248745968755546),
(('Y', '4', (2, 2, 2)), 0.13186898759523)
…
…
(('Z', '10', (0, 0, 0)), -0.15880353875480954),
(('W', '10', (5, 5, 5)), -0.17251241903947837)]

Task 2:

In this task, given a query gesture object, we propose different ways to find its top 10 similar gestures for the following 7 user options: Dot Product, PCA, SVD, NMF, LDA, Edit Distance and DTW. The results below show the **top 10 similar gestures** of 3 gesture files (**1.csv, 260.csv** and **589.csv**) for different user options using the **TF model**.

User option 1: Dot Product

In this user option, the top 10 similar gestures to the query object are identified using a dot product between the TF or TF-IDF vector representation (as chosen by the user) to compute the similarity between the query object and every other object in the dataset and return the top 10 gestures with highest similarity values. If two objects have the same similarity, those two objects will be returned in any order.

Dot Product Similarity Results: Model = TF and p = 4

a. ***Gesture File: 1.csv***

('1', 0.002409393491124248)
('277', 0.0019471153846153826)
('7', 0.0018902551775147904)
('269', 0.0018634430473372765)
('251', 0.0018181397928994055)
('564', 0.001797191295546561)
('253', 0.0017816742081447972)
('257', 0.0017797707100591703)
('5', 0.0017761145104895082)
('275', 0.001775090144230771)

b. ***Gesture File: 260.csv***

('260', 0.003507812499999979)
('266', 0.002482077205882364)
('255', 0.0023968749999999954)
('267', 0.0023331801470588293)
('251', 0.00229447115384616)
('273', 0.002257291666666665)
('253', 0.0022233455882353301)
('256', 0.0022182617187500044)
('263', 0.002197482638888883)
('261', 0.002174879807692311)

c. ***Gesture File: 589.csv***

('589', 0.002816162109374959)
('21', 0.0021367938701923007)
('579', 0.0020901396780303077)
('578', 0.0020052083333333367)
('580', 0.00200080422794116 8)
('569', 0.001983351934523806)
('269', 0.001963641826923079)

('277', 0.0019596354166666703)
('18', 0.0019497445913461587)
('249', 0.001918945312500004)

User option 2: PCA

The top k latent topics identified from user option PCA in Task 1 are used to represent each gesture object in the new latent feature space of k dimensions. This transformed and reduced dimension data in the form of an *n x k* matrix is used to extract the top 10 similar gestures for a query gesture object. Essentially each gesture object represented as a k-length vector is compared with every other gesture object in the dataset of k-length vector using **cosine similarity**, and ranked in decreasing order of their similarity score. Then the top 10 similar gestures are returned to the user.

PCA Similarity Results: Model = TF and p = 4

a. ***Gesture File: 1.csv***

('1', 1.0)
('2', 0.9799090953314009)
('30', 0.8353261086446123)
('24', 0.8096011660841643)
('23', 0.801269602181702)
('4', 0.7937293088598424)
('12', 0.7832288276656684)
('27', 0.7739431495490829)
('14', 0.7121421343776538)
('15', 0.7084825369337047)

b. ***Gesture File: 260.csv***

('260', 1.0)
('266', 0.913576805262215)
('256', 0.8863254561940767)
('564', 0.8816183218562861)
('254', 0.8804363954998091)
('267', 0.7818935626800836)
('263', 0.7201034767877161)
('255', 0.6881382075699799)
('265', 0.6863162740490135)
('272', 0.6827749686526893)

c. ***Gesture File: 589.csv***

('589', 1.0)

('578', 0.9903343612186699)
('580', 0.9606083422973128)
('582', 0.958583732352825)
('583', 0.9411064819812593)
('588', 0.9038469722807038)
('581', 0.855973362943944)
('585', 0.7970347103547885)
('584', 0.7950068611942187)
('586', 0.7907757721422427)

User option 3: SVD

The top k latent topics identified from user option SVD in Task 1 are used to represent each gesture object in the new latent feature space of k dimensions. This transformed data in the form of an *n x k* matrix in the new feature space is used to extract the top 10 similar gestures for a query gesture object. Essentially each gesture object represented as a k-length vector is compared with every other gesture object in the dataset of k-length vector using **cosine similarity**, and ranked in decreasing order of their similarity score. Then the top 10 similar gestures are returned to the user.

SVD Similarity Results: Model = TF and p = 4

a. ***Gesture File: 1.csv***

('1', 0.9999999999999999)
('269', 0.9984991266428316)
('2', 0.9976954614563969)
('582', 0.9955348700714954)
('23', 0.9940964167228085)
('30', 0.9936127505870785)
('559', 0.9924282528572116)
('12', 0.99231515097204)
('4', 0.9893945776413919)
('588', 0.9848387252130177)

b. ***Gesture File: 260.csv***

('260', 1.0000000000000002)
('266', 0.997231690253995)
('256', 0.9893476092271154)
('267', 0.9821411556895674)
('254', 0.9733344006189546)
('564', 0.9569741476405244)
('272', 0.9549852646724662)
('263', 0.9515381102352308)

('271', 0.9514892461352306)
('265', 0.9473199115302173)

c. Gesture File: 589.csv

('589', 1.0000000000000002)
('578', 0.9999553549069558)
('583', 0.9981680090178018)
('587', 0.9968598855634754)
('580', 0.9965811562404359)
('12', 0.9937712392496955)
('582', 0.9890258698470448)
('588', 0.987813519462978)
('581', 0.9869611759607726)
('585', 0.9852591169034818)

User option 4: NMF

NMF Similarity Results: Model = TF and p = 4

a. Gesture File: 1.csv

('1', 0.9999999999999999)
('2', 0.9890494523756237)
('269', 0.9766151475885336)
('23', 0.9765743775892604)
('12', 0.9669712858065767)
('24', 0.9654128362120535)
('27', 0.9603242363857959)
('559', 0.9511148254127958)
('9', 0.9319680364804637)
('30', 0.930516152935136)

b. Gesture File: 260.csv

('260', 1.0)
('266', 0.9998544061549104)
('256', 0.9886669244762796)
('267', 0.9688953692127934)
('564', 0.9399259615671726)
('254', 0.9322674871202615)
('272', 0.9212801818581107)
('577', 0.9062767812340768)
('264', 0.896837625034678)

('572', 0.8874072689123464)

c. Gesture File: 589.csv

('589', 1.0000000000000002)
('578', 0.9998295016717704)
('582', 0.9874776201693438)
('583', 0.987128766134309)
('587', 0.9706697172697337)
('580', 0.9659392624439173)
('269', 0.9565841526281884)
('12', 0.9554949784126613)
('588', 0.9514791410323964)
('581', 0.9472006932210606)

User option 5: LDA

Here we consider the generative LDA model for similarity scores. The matrix represents the corpus. Each row of the matrix can be seen as a document. Task 1 gets an N * M matrix and LDA operates on it to give a resultant N * K representation of the documents in terms of topics. We use this representation as input to KL divergence to calculate the similarities. KL divergence can be seen as distance and hence we do 1/(1 + score)

LDA Similarity Results: for model = tf and p = 4

a. Gesture File: 1.csv

('1', 1.0)
('26', 0.9999998846025684)
('30', 0.9999998720205934)
('272', 0.9999996274406369)
('24', 0.999999517422657)
('20', 0.9999993659704859)
('22', 0.9999990250135637)
('577', 0.9999981096391041)
('31', 0.9999981038874574)
('29', 0.9999980433047639)

b. Gesture File: 260.csv

('260', 1.0)
('268', 0.9999999973707943)
('276', 0.9999999889722012)
('571', 0.9999999876415925)
('256', 0.9999999753396368)

('254', 0.9999999599319342)
('266', 0.9999999227425977)
('252', 0.9999998751934274)
('273', 0.9999998644689229)
('255', 0.999999801140045)

c. Gesture File 589.csv:

('589', 1.0)
('14', 0.9999999988359645)
('262', 0.9999999880384879)
('587', 0.9999999563428742)
('574', 0.9999999385892688)
('559', 0.9999999328557291)
('581', 0.9999999296771568)
('19', 0.9999999084333469)
('12', 0.9999998930949417)
('572', 0.9999998717084513)

In addition to this, let us consider the N * K representation obtained by applying LDA to the original N * M matrix for a few documents. Here K = 4, i.e. each document is represented in terms of 4 topics.

File Name:	**Contribution of topics:**
572	[**0.61188661** 0.12937113 0.12937113 0.12937113]
253	[**0.61406986** 0.12864338 0.12864338 0.12864338]
561	**[0.61418431** 0.12860523 0.12860523 0.12860523]
8	**[0.61172587** 0.12942471 0.12942471 0.12942471]
15	[**0.60643636** 0.13118788 0.13118788 0.13118788]
585	[**0.61070015** 0.12976662 0.12976662 0.12976662]
560	[**0.61306688** 0.12897771 0.12897771 0.12897771]
28	[**0.60872619** 0.1304246 0.1304246 0.1304246]
268	[**0.61366583** 0.12877806 0.12877806 0.12877806]
272	[**0.60931159** 0.13022947 0.13022947 0.13022947]
579	[**0.61473032** 0.12842323 0.12842323 0.12842323]
9	[**0.61253722** 0.12915426 0.12915426 0.12915426]
17	[**0.60857125** 0.13047625 0.13047625 0.13047625]
580	[**0.61270785** 0.12909738 0.12909738 0.12909738]
274	[**0.61282255** 0.12905915 0.12905915 0.12905915]
270	[**0.61320351** 0.12893216 0.12893216 0.12893216]
252	[**0.61345785** 0.12884738 0.12884738 0.12884738]
583	[**0.61135446** 0.12954851 0.12954851 0.12954851]
588	[**0.61289407** 0.12903531 0.12903531 0.12903531]
14	[**0.61215696** 0.12928101 0.12928101 0.12928101]
273	[**0.61395461** 0.1286818 0.1286818 0.1286818]
258	[**0.61318646** 0.12893785 0.12893785 0.12893785]
575	[**0.61279848** 0.12906717 0.12906717 0.12906717]

587 [**0.61227742** 0.12924086 0.12924086 0.12924086]
562 [**0.61419655** 0.12860115 0.12860115 0.12860115]
275 [**0.6141041** 0.12863197 0.12863197 0.12863197]
22 [**0.60905139** 0.1303162 0.1303162 0.1303162]

We see that in explaining each of these files, the first topic is contributing maximum i.e. around 60 - 62%. The topics 2, 3 and 4 have relatively lesser contribution in explaining the files. As a result of this, the reduced representations of these files are similar.

We observe the same behaviour for greater values of K. For example, below are the topic representations for K = 10:

568 [0.0500027 0.0500027 0.0500027 0.0500027 **0.54997572** 0.0500027 0.0500027 0.0500027 0.0500027 0.0500027]
13 [0.05000253 0.05000253 0.05000253 0.05000253 **0.54997723** 0.05000253 0.05000253 0.05000253 0.05000253 0.05000253]
26 [0.05000396 0.05000396 0.05000396 0.05000396 **0.5499644** 0.05000396 0.05000396 0.05000396 0.05000396 0.05000396]
586 [0.05000373 0.05000373 0.05000373 0.05000373 **0.54996644** 0.05000373 0.05000373 0.05000373 0.05000373 0.05000373]
576 [0.05000299 0.05000299 0.05000299 0.05000299 **0.54997306** 0.05000299 0.05000299 0.05000299 0.05000299 0.05000299]
567 [0.05000255 0.05000255 0.05000255 0.05000255 **0.54997705** 0.05000255 0.05000255 0.05000255 0.05000255 0.05000255]
261 [0.05000244 0.05000244 0.05000244 0.05000244 **0.54997806** 0.05000244 0.05000244 0.05000244 0.05000244 0.05000244]
254 [0.05000274 0.05000274 0.05000274 0.05000274 **0.54997531** 0.05000274 0.05000274 0.05000274 0.05000274 0.05000274]
5 [0.05000267 0.05000267 0.05000267 0.05000267 **0.54997595** 0.05000267 0.05000267 0.05000267 0.05000267 0.05000267]
16 [0.0500026 0.0500026 0.0500026 0.0500026 **0.54997664** 0.0500026 0.0500026 0.0500026 0.0500026 0.0500026]
249 [0.05000243 0.05000243 0.05000243 0.05000243 **0.54997816** 0.05000243 0.05000243 0.05000243 0.05000243 0.05000243]

Here as well, we see that there is one topic (topic 5) that is contributing the most in explaining the file. The percentage of the highest contributor has dropped from around 61 to around 55 because of introducing more topics and those doing some of the explanation. Thus, for our dataset, LDA representation is giving us one dominant topic and an almost even distribution in all the other topics and this is true for all the files. Additionally, it is the same topic that is contributing the most in explaining all the files.

User option 6: Edit Distance

Up until now, we have considered that each object, or a in our case, each gesture file is represented using a list of words(tf or tf-idf vectors), or after the above-mentioned dimensionality reduction techniques, some *k* latent semantics. Now, to compare any two files, we used

standard similarity metrics like dot product and cosine similarity, which take into account the angular separation between two objects.

The issue with these metrics is that they fail to take into consideration the fact that this is a time-series data we are working with. This means the order in which each word appears should be of significant importance. Along with the order, there could be cases where instances of stretched or shifted data are compared. Ideally, a metric should be able to recognize that. Edit distance is one such metric that works well with temporal data. It calculates how many operations or edits one has to make to convert one string into another.

To understand it better, we can assume the symbolic representation of each sensor for a gesture file, for every component, is a string. We will compare the same sequence in both files, and finally return the average of edit distances for all the sequences. The figure below shows an example of two sequences we might use for comparison:

[[6, 5, 5], [5, 5, 5], [5, 4, 4], [4, 4, 4], [4, 3, 3], [3, 3, 3], [3, 3, 3], [3, 3, 2], [2, 1, 2], [2, 2, 2], [2, 2, 2], [2, 2, 2], [2, 3, 3], [3, 3, 3], [3, 3, 4], [4, 4, 4], [4, 4, 5], [5, 5, 5], [5, 5, 5]]
Sequence 1: File 1.wrd, component = X, sensor_id = 0

[[6, 6, 6], [6, 5, 5], [5, 5, 5], [5, 5, 5], [5, 4, 4], [4, 4, 4], [4, 3, 3], [3, 3, 3], [3, 3, 3], [3, 3, 3], [3, 3, 2], [2, 2, 2], [2, 2, 2], [2, 2, 2], [2, 2, 2], [2, 2, 2], [2, 1, 2], [2, 2, 2], [2, 3, 3]]
Sequence 2: File 2.wrd, component = X, sensor_id = 0

Here, if each sequence is a string, then each of these tuples(for ex: [4, 4, 4]) is a word. Now, if our data has 4 components(W, X, Y, Z) and 20 sensors, then we will end up comparing 80 pairs of sequences. As we are working on symbols, we have considered the following cost function for each edit operation.

Cost of inserting a word = 3
We have assumed each symbol has equal importance.
For example: inserting a word like [4, 4, 4] is equivalent to adding 3 values in the original data.

Cost of deleting a word = 3
We have assumed each symbol has equal importance.
For example: deleting a word like [4, 4, 4] is equivalent to removing 3 values in the original data.

Cost of substituting a word = Number of different symbols in the word
We have assumed each symbol has equal importance.
For example: replacing a word like [4, 4, 4] with a word like [4, 3, 3] is equivalent to replacing 2 different symbols or values in the original data.

We used the **dynamic programming** algorithm that Prof. Candan taught in his lectures and obtained a distance measure for every pair of sequences. Finally, we took the average of these

values and considered it to be our mean edit distance between two gesture files. The following figure shows a snippet of the code used to measure edit distance:

```
for row in range(1, rows):
        dist[row][0] = row * 3 // For every delete

for col in range(1, cols):
        dist[0][col] = col * 3 // For every insert

for row in range(1, rows):
        for col in range(1, cols):
                if s[row - 1] == t[col - 1]:
                        cost = 0 // Cost of substitution
                else:
                        cost = 0
                        for i in range(len(s[row - 1])):
                        if s[row - 1][i] != t[col - 1][i]:
                        cost += 1 // Cost of substitution

                dist[row][col] = min(dist[row - 1][col] + 3,  # deletes
                                dist[row][col - 1] + 3,  # inserts
                                dist[row - 1][col - 1] + cost)  # substitution
```

Figure: Edit distance dynamic programming algorithm

As we were asked for a similarity measure, we used a simple transformation to calculate the similarity:

$$Similarity = \frac{1}{1 + Distance}$$

This ensures the same files will be given a similarity score of 1.

We ran this code on 3 query files, namely, 1.csv, 260.csv, and 589.csv by working directly with their .wrd files. The following shows the results obtained from each:

a. Gesture file: 1.csv
   [['1' '1.0']
   ['11' '0.043010752688172046']
   ['12' '0.042105263157894736']
   ['24' '0.04171011470281543']
   ['2' '0.04153686396677051']
   ['29' '0.041109969167523124']
   ['6' '0.040774719673802244']
   ['30' '0.0404244567963618']
   ['20' '0.039761431411530816']

['272' '0.03950617283950617']]

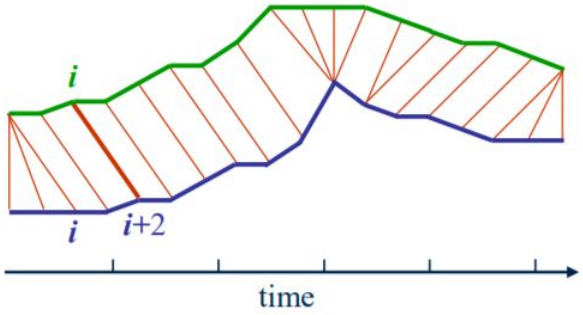
b. Gesture file: 260.csv
[['260' '1.0']
['256' '0.030383592859855677']
['266' '0.030041306796845663']
['258' '0.02867383512544803']
['573' '0.028612303290414875']
['254' '0.028388928317955996']
['255' '0.028358738036157388']
['253' '0.028040658955485454']
['250' '0.027981811822315496']
['572' '0.02746309646412633']]

c. Gesture file: 589.csv
[['589' '1.0']
['582' '0.035650623885918005']
['583' '0.03463203463203463']
['580' '0.03454231433506045']
['584' '0.033277870216306155']
['566' '0.033236393851267136']
['585' '0.03307151715584952']
['587' '0.03191065017949741']
['586' '0.03180914512922465']
['588' '0.031658092599920855']]

User option 7:

DTW Distance:

The other algorithm that works well with the linear sequence is the Dynamic Time Warping. DTW algorithm tries to find an optimal alignment between the time-series sequence to give a more optimal similarity even if they are out of order in the time-axis. This algorithm works well in matching sequences that are either stretched or shifted from each other.



The way the DTW algorithm works is by finding the minimum cost of transforming the data point in the sequence into the other sequence. For doing this it takes into account the cost of transforming a symbol in sequence 1 into another symbol in sequence 2.

The input to the DTW algorithm are two sequences of symbols and a cost matrix that consists of the cost of transforming symbols into another symbol. The output is the cost of converting sequence 1 to sequence 2.

In this task we are using the window descriptor that we had generated from task 0a as the sequence of symbols and the quantised amplitude of the window descriptor as its cost. Now for the comparison for two gestures we compare the sequence of window descriptor from a particular component and sensor with the same component and sensor window descriptor of the other sequence. Now the cost of changing a symbol into another symbol is the absolute value of the difference in amplitude between the two symbols.

Cost = abs(Amplitude of symbols in sequence 1 - amplitude of symbols in sequence 2)

Example:

Cost of changing a symbol [3,3,3] with [4,4,4] having amplitude 3 and 4 is absolute difference in there amplitude i.e 1.

There are some constraints on the wrap path for DTW algorithm and they are -

- **Monotonicity**: The symbols in the sequence should be ordered with respect to the time. A symbol in time t should occur after a symbol at time (t-1)
- **Continuity**: The path advances one step at a time and is limited to its neighbouring states.
- **Boundary condition**: The path starts from the bottom left and ends at the top right corner.

Input to the algorithm:

[[6, 5, 5], [5, 5, 5], [5, 4, 4], [4, 4, 4], [4, 3, 3], [3, 3, 3], [3, 3, 3], [3, 3, 2], [2, 1, 2], [2, 2, 2], [2, 2, 2], [2, 2, 2], [2, 3, 3], [3, 3, 3], [3, 3, 4], [4, 4, 4], [4, 4, 5], [5, 5, 5], [5, 5, 5]]
Sequence 1: File 1.wrd, component = X, sensor_id = 0

[0.935410833780014, 0.9050157256503621, 0.5742150256241504, 0.4088146756110446, -0.136271558537014, -0.408814675611044, -0.408814675611044, -0.574215025624150..]
Cost 1: File 1.wrd, component=X, sendor_id = 0

[[6, 6, 6], [6, 5, 5], [5, 5, 5], [5, 5, 5], [5, 4, 4], [4, 4, 4], [4, 3, 3], [3, 3, 3], [3, 3, 3], [3, 3, 3], [3, 3, 2], [2, 2, 2], [2, 2, 2], [2, 2, 2], [2, 2, 2], [2, 2, 2], [2, 1, 2], [2, 2, 2], [2, 3, 3]]
Sequence 2: File 2.wrd, component = X, sensor_id = 0

[0.935410833780014, 0.9050157256503621, 0.5742150256241504, 0.4088146756110446, -0.136271558537014, -0.408814675611044, -0.408814675611044, -0.574215025624150..]
Cost 2: File 2.wrd, component=X, sendor_id = 0

Algorithm:

We are using the dynamic programming approach to calculate the DTW distance between two sequences that Prof. Candan had taught during his lectures.

```
cost1= Amplitude of Symbols in sequence 1
cost2= Amplitude of Symbols in sequence 2
row= length of sequence 1
col = length of sequence 2
        dp[row][col] = Infinity
        dp[row - 1][0] = 0 # Starting from left bottom
        For i in range row-1 -> 1:
                For j in range 1 -> Col:
                        cost= 0
                        If symbol1[i] not eq symbol2[j]:
                                Cost = abs(cost1[i] - cost2[j])
                        Dp[i][j] = cost + min (dp[i + 1][j], dp[i][j - 1], dp[i + 1][j
                - 1])
        Return dp[0][col-1]
```

If both the sequences contain the same symbols the DTW distance will be 0. In the similar way all the sequences belonging to each sensor_id and components are calculated and the mean is calculated and stored. Finally top 10 gesture files are returned with their corresponding cost. Here two gesture files are more similar to each other if the DTW distance is minimum among all the other gesture files. Hence similarity from the DTW distance is calculated as follows -

$$Similarity(fi,fj) = \frac{1}{1 + DTW(fi,fj)}$$

Hence the similarity score with itself is always 1 as the cost is 0. The DTW distance was run on 3 query files, namely, 1.csv, 260.csv, and 589.csv. The following shows the results based on similarity obtained for each query file:

1.csv

```
[['1' '1.0']
 ['11' '0.22618185172345645']
 ['577' '0.22574310038365336']
 ['4' '0.22447832403506823']
 ['28' '0.22391558711006357']
 ['19' '0.21488659135401758']
 ['24' '0.2135523552334951']
 ['20' '0.21031763080424093']
 ['12' '0.20858916458348817']
 ['2' '0.20800465148767203']]
```

260.csv

```
[['260' '1.0']
 ['266' '0.20250274022277398']
 ['572' '0.20110541098102122']
 ['258' '0.19571311958230622']
 ['575' '0.19442353947640362']
 ['30' '0.1897301923990446']
 ['265' '0.187642397392342']
 ['573' '0.18697700502346687']
 ['566' '0.18696914288767497']
 ['254' '0.1858557258165932']]
```

589.csv

```
[['589' '1.0']
 ['582' '0.22663889046745245']
 ['585' '0.22454710291735416']
 ['566' '0.22159815982718076']
 ['580' '0.21181399622110292']
 ['583' '0.21179055718498915']
 ['581' '0.20644896023518206']
 ['586' '0.19850884525646967']
 ['8' '0.19578906280034092']
 ['584' '0.19018559113973796']]
```

Thus, we summarize and compare the results obtained via all the seven methods in the table shown below. We can observe that DTW and Edit Distance sometimes give better results

because of their ability to capture temporal data. For some files, we observe that PCA, SVD, NMF do not perform that well because there is some information loss when we reduce the dimensions.

	Rank	DOT	PCA	SVD	NMF	LDA	ED	DTW
1.csv	1 2 3 4 5 6 7 8 9 10	1 277 7 269 251 564 253 257 5 275	1 2 30 24 23 4 12 27 14 15	1 269 2 582 23 30 559 12 4 588	1 2 269 23 12 24 27 559 9 30	1 26 30 272 24 20 22 577 31 29	1 11 12 24 2 29 6 30 20 272	1 11 577 4 28 19 24 20 12 2
260.csv	1 2 3 4 5 6 7 8 9 10	260 266 255 267 251 273 253 256 263 261	260 266 256 564 254 267 263 255 265 272	260 266 256 267 254 564 272 263 271 265	260 266 256 267 564 254 272 577 264 572	260 268 276 571 256 254 266 252 273 255	260 256 266 258 573 254 255 253 250 572	260 266 572 258 575 30 265 573 566 254
589.csv	1 2 3 4 5 6 7 8 9 10	589 21 579 578 580 569 269 277 18 249	589 578 580 582 583 588 581 585 584 586	589 578 583 587 580 12 582 588 581 585	589 578 582 583 587 580 269 12 588 581	589 14 262 587 574 559 581 19 12 572	589 582 583 580 584 566 585 587 586 588	589 582 585 566 580 583 581 586 8 584

Task 3: Latent Gesture Discovery Tasks

In this task, we create gesture-gesture similarity matrices using user selected options(*Dot-product, PCA, SVD, NMF, LDA, Edit Distance, DTW*) and perform *SVD* and *NMF(dimensionality reduction techniques)* on the similarity matrices. We then report the top-*p*

principle components underlying the user-selected option of a gesture-gesture similarity matrix. The report contains the gestures along with their degree of membership to the top p principle components in non-increasing order in the form of json files called *phase_3_task3_SVD_output.json and phase_3_task3_NMF_output.json* respectively.

All the gesture-gesture similarity matrices are saved in a *.csv* format with the following name: *task3_{metric}_sim_matrix.csv* where metric could be *PCA, SVD, DTW*, etc. as per the user selected option.

Gesture-Gesture similarity matrix:

The gesture-gesture similarity matrix gives us a view of the similarity scores between different gestures. We compute the similarity matrix using the dot product between the top-k latent semantics/topics generated in the previous tasks.

The gesture-gesture similarity score might be a good factor to see the similarity between two gestures, but it isn't always the case. For example, In case of dot product, the similarity score of a vector multiplying with itself may not yield a high score if the amplitudes of each dimension are already low. So a high/low score in case of dot product may not correlate at all to the similarity.

User Option 1: Dot Product

The figure below shows a snippet of the similarity matrix generated by this algorithm.

	Unnamed: 0	1	10	...	7	8	9
0	1	0.002285	0.000206	...	0.000215	0.000256	0.000239
1	10	0.000206	0.001146	...	0.000233	0.000291	0.000224
2	11	0.000316	0.000251	...	0.000218	0.000221	0.000244
3	12	0.000253	0.000279	...	0.000172	0.000242	0.000284
4	13	0.000214	0.000261	...	0.000237	0.000232	0.000217
..	...	...	...	...	...	...	...
88	589	0.000235	0.000283	...	0.000201	0.000229	0.000232
89	6	0.000295	0.000256	...	0.000214	0.000220	0.000312
90	7	0.000215	0.000233	...	0.000819	0.000212	0.000231
91	8	0.000256	0.000291	...	0.000212	0.001775	0.000282
92	9	0.000239	0.000224	...	0.000231	0.000282	0.001357

User Option 2: PCA

The figure below shows a snippet of the similarity matrix generated by this algorithm.

	Unnamed: 0	1	10	...	7	8	9
0	1	1.000000	0.371205	...	0.587917	-0.331940	0.290013
1	10	0.371205	1.000000	...	0.843611	0.051943	0.742132
2	11	0.210631	0.891394	...	0.898779	-0.250715	0.910513

3	12	0.476284	0.870117	...	0.640115	0.337833	0.391081
4	13	0.390184	0.994970	...	0.814036	0.120906	0.682443
..	...	...	...	...	...	...	...
88	589	0.574470	0.895588	...	0.939706	-0.321648	0.864015
89	6	0.627365	0.684569	...	0.921461	-0.638197	0.892406
90	7	0.587917	0.843611	...	1.000000	-0.472418	0.923894
91	8	-0.331940	0.051943	...	-0.472418	1.000000	-0.586680
92	9	0.290013	0.742132	...	0.923894	-0.586680	1.000000

User Option 3: SVD

The figure below shows a snippet of the similarity matrix generated by this algorithm.

	Unnamed: 0	1	10	...	7	8	9
0	1	1.000000	0.998564	...	0.995752	0.958765	0.993389
1	10	0.998564	1.000000	...	0.994878	0.966200	0.994114
2	11	0.988331	0.988473	...	0.997250	0.923333	0.993770
3	12	0.993991	0.994940	...	0.986363	0.974442	0.980687
4	13	0.997587	0.996587	...	0.995083	0.957063	0.989755
..	...	...	...	...	...	...	...
88	589	0.998784	0.996756	...	0.998394	0.944923	0.996901
89	6	0.984097	0.977145	...	0.991484	0.893132	0.988719
90	7	0.995752	0.994878	...	1.000000	0.935707	0.998328
91	8	0.958765	0.966200	...	0.935707	1.000000	0.934928
92	9	0.993389	0.994114	...	0.998328	0.934928	1.000000

User Option 4: NMF

The figure below shows a snippet of the similarity matrix generated by this algorithm.

	Unnamed: 0	1	10	...	7	8	9
0	1	1.000000	0.995140	...	0.988178	0.960821	0.986439
1	10	0.995140	1.000000	...	0.997702	0.936802	0.997550
2	11	0.985397	0.997003	...	0.999399	0.910825	0.997903
3	12	0.997128	0.989449	...	0.980019	0.973847	0.978103
4	13	0.995348	0.998237	...	0.997254	0.935720	0.992923
..	...	...	...	...	...	...	...
88	589	0.990939	0.997569	...	0.998827	0.916192	0.995884
89	6	0.982539	0.993366	...	0.997551	0.894299	0.994561
90	7	0.988178	0.997702	...	1.000000	0.912540	0.997269
91	8	0.960821	0.936802	...	0.912540	1.000000	0.917766
92	9	0.986439	0.997550	...	0.997269	0.917766	1.000000

User Option 5: LDA

The figure below shows a snippet of the similarity matrix generated by this algorithm.

	Unnamed: 0	1	10	...	7	8	9

0	1	1.000000	0.993868	...	0.984886	0.997897	0.997122
1	10	0.993809	1.000000	...	0.997936	0.998927	0.999379
2	11	0.998894	0.997939	...	0.991814	0.999838	0.999577
3	12	0.999382	0.997112	...	0.990254	0.999554	0.999162
4	13	0.989183	0.999347	...	0.999602	0.996605	0.997456
..	...	...	...	...	...	...	...
88	589	0.998145	0.998729	...	0.993467	0.999991	0.999884
89	6	0.999332	0.997217	...	0.990445	0.999595	0.999218
90	7	0.984715	0.997933	...	1.000000	0.993900	0.995058
91	8	0.997883	0.998930	...	0.993929	1.000000	0.999939
92	9	0.997100	0.999381	...	0.995077	0.999938	1.000000

User Option 6: Edit Distance

The figure below shows a snippet of the similarity matrix generated by this algorithm.

	1	10	11	...	7	8	9
1	1.000000	0.025517	0.045873	...	0.019227	0.037790	0.027474
10	0.025517	1.000000	0.025333	...	0.021771	0.029169	0.027275
11	0.045873	0.025333	1.000000	...	0.019212	0.037451	0.026921
12	0.044512	0.026461	0.044866	...	0.019255	0.041372	0.027072
13	0.017417	0.020969	0.017773	...	0.022718	0.018547	0.018856
..	...	...	...	...	...	...	...
589	0.029157	0.027561	0.029295	...	0.018997	0.031233	0.025734
6	0.042574	0.025054	0.046276	...	0.018102	0.037066	0.026210
7	0.019227	0.021771	0.019212	...	1.000000	0.018644	0.022296
8	0.037790	0.029169	0.037451	...	0.018644	1.000000	0.025321
9	0.027474	0.027275	0.026921	...	0.022296	0.025321	1.000000

This matrix is then given as input to SVD and NMF where $p = 4$, and the desired output is generated. Here, we return 2 json files which contain the contributions of each gesture file to a latent semantic. The issue here is that they are not sorted on the basis of their contributions, hence, along with these json files, we generate 2 csv files which contain this same information, but sorted on the basis of their contributions. This additional file will be helpful for future tasks.

User Option 7: DTW Distance

Below is the representation of gesture-gesture similarity matrix formed using DTW sim

	Unnamed: 0	1	10	...	7	8	9
0	1	1.000000	0.220922	...	0.251589	0.217785	0.237733
1	10	0.220922	1.000000	...	0.185806	0.259623	0.171788
2	11	0.279223	0.185042	...	0.206855	0.207746	0.185244
3	12	0.286634	0.221205	...	0.210427	0.235825	0.227590

4	13	0.156584	0.229982	...	0.184116	0.202092	0.127769
..	...	...	...	...	...	...	...
88	589	0.197660	0.225671	...	0.142702	0.230749	0.174212
89	6	0.243605	0.152129	...	0.143346	0.182007	0.149252
90	7	0.251589	0.185806	...	1.000000	0.187020	0.229673
91	8	0.217785	0.259623	...	0.187020	1.000000	0.181041
92	9	0.237733	0.171788	...	0.229673	0.181041	1.000000

Performing SVD and NMF on the gesture-gesture similarity matrix:

When we send gesture-gesture similarity matrix as an input to SVD or NMF, we are treating the input as gestures represented in terms of other gestures. Unlike the previous tasks, the vector space here has gestures as dimensions and gestures as data points as well.

When we reduce the dimensions by passing the similarity matrix to SVD or NMF, we essentially are trying to represent gestures in terms of latent-gestures. So the transformed matrix can be seen as all the gestures represented in terms of multiple gestures that are linear combinations of original gestures.

For example, for User Option 6, the SVD decomposition is:

	1	10	11	12	..
1	0.116011991906376	0.091816624497169	0.116642531868973	0.11652610987216	..
2	0.05768204855695	-0.087940395125237	0.043124600786575	0.045884550295638	..
3	-0.060541892517494	0.076209396503856	-0.022015872877835	0.061379176028236	..
4	-0.044814683678579	-0.166163171147193	0.083426471314208	-0.048584082487586	..

For example, for User Option 6, the NMF decomposition is:

	1	10	11	12	...
1	0.106321754899528	0.016980300790439	0.101109800768204	0.104112813846494	..
2	0.007394580668262	0.182452509699679	0.004099093428164	0.022518394753559	..
3	0.220898293154981	0	0.231876355172628	0.212611851384337	..
4	0.026083611643144	0.153128979988359	0.025751403004541	0.025152359132304	..

This linear combination of original gestures is then sorted for each latent-gesture semantic and stored in a json file.

Task 4: Latent Gesture Clustering and Analysis Tasks

Task 4a:

This task uses the output we get from Task 3a based on the user options. Here the value of p is the same as that decided in task 3a. The output is of the shape (p X Number of gestures) and the value is the contribution of the gesture to that latent semantic. Now in this task we group the gestures based on its highest contribution to the latent semantic. For example if a gesture has its highest contribution to latent semantic p1 then the file is grouped under p1, the result of this task will basically group the gestures based on its highest contribution to the p-semantics in other words a gesture contributing its highest to any latent semantic will be grouped under it.

Input:

Here the row are the gesture and column are the p-latent semantic and the value are the contribution of the gesture to the latent semantic

	1	10	11	12	13	14	15	16	17	18	19	2	20
1	0.661256	0.669529	0.679777	0.743294	0.48666	0.487033	0.026082	0.450882	0	0.526949	0.578582	0.631632	0.07931
2	0.001827	0.037269	0.041746	0.010825	0	0.001684	0.063979	0.021272	0	0.006453	0.138588	0.146291	0.00315
3	0.587089	0.593295	0.583799	0.441801	0.877764	0.47993	0	0.927344	0	0.824485	0.469428	0.63443	0.012238
4	0.167259	0.090559	0.029554	0.211206	0.095527	0.591268	0.019586	0.121622	0.897377	0	0.460825	0.000172	0.894365

Fig: Partial Input image for task 4A

Algorithm:

```
Map of  p => set()
 For each file_number, data in Transpose(Input):
        index,value= Find_Highest_value_With_index(data)
        Add file_number to Map for that p
Return Map
```

Working:

Here we are grouping all the gesture files based on its highest contribution to the latent semantic. Here the contribution of the gesture file to a latent semantic can be interpreted in two ways.

1. Highest contribution based on absolute value(ignoring sign):
   In this we cluster the gesture based on the magnitude of its contribution to any latent semantic. Here the sign of contribution to a latent semantic is ignored. A gesture that has contribution to 0.6 to p1(latentent semantic) and -0.7 to p2 is clustered under p2 based on the magnitude ignoring the direction/sign.

2. Highest contribution based on the value:

In this we cluster the gesture based on its contribution calculated based on the value. Here a gesture that has contribution to -0.6 to p1(latentent semantic) and 0.1 to p2 is clustered under p2 based on the value. This was done for the comparison and see the benefits and drawbacks of grouping it based on value and considering sign

Below are the results of the clustering performed on different input matrix obtained from task 3a

Taking absolute value for finding maximum contribution of a gesture to an Latent Feature

Dot product gesture-gesture similarity matrix results:

1: {'276', '584', '27'},
2: {'26', '22', '20', '23', '17', '24', '28'},
3: { '1', '10', '11', '12', '13', '14', '16', '18', '19', '2', '21',
'249', '25', '250', '251', '252', '253', '254', '255', '256', '257',
'258', '259', '260', '261', '262', '263', '264', '265', '266', '267',
'268', '269', '270', '271', '272', '273', '274', '275', '277', '278',
'279', '29', '3', '30', '31', '4', '5', '559', '560', '561', '562',
'563', '564', '565', '566', '567', '568', '569', '570', '571', '572',
'573', '574', '575', '576', '577', '578', '579', '580', '581', '582',
'583', '585', '586', '587', '588', '589', '6', '7', '8', '9'},
4: {'15'}

PCA gesture-gesture similarity matrix results

{ 1: { '18', '19', '21', '24', '249', '250', '251', '252', '253', '254',
'255', '256', '257', '258', '259', '260', '261', '262', '263', '264',
'265', '266', '268', '269', '271', '273', '277', '279', '559', '561',
'562', '564', '565', '574', '576', '579', '582', '585', '586', '588',
'6', '7', '9'},
2: { '10', '12', '13', '14', '20', '22', '26', '270', '276', '29', '566',
'568', '569', '570', '571', '573', '578', '8'},
3: { '1', '11', '16', '17', '23', '272', '274', '278', '28', '560', '563',
'567', '575', '577', '580', '581', '583', '587'},
4: { '15', '2', '25', '267', '27', '275', '3', '30', '31', '4', '5',
'572', '584', '589'}}

SVD gesture-gesture similarity matrix results:

{ 1: { '1', '10', '11', '12', '13', '16', '18', '2', '21', '249', '251',
'252', '255', '256', '257', '259', '261', '262', '264', '267', '268',
'269', '270', '272', '273', '274', '275', '277', '278', '279', '3',
'4', '5', '559', '560', '561', '562', '563', '564', '565', '566',
'567', '568', '569', '570', '571', '572', '573', '574', '575', '576',

'577', '578', '579', '580', '582', '583', '586', '587', '588', '589', '6', '7', '9'},
2: { '14', '17', '19', '20', '22', '23', '24', '25', '250', '253', '254', '258', '26', '260', '263', '265', '266', '271', '28', '29', '30', '31', '585', '8'},
3: {'276', '581', '584', '27'},
4: {'15'}}

NMF gesture-gesture similarity matrix results:

{ 1: { '1', '10', '11', '12', '13', '16', '18', '2', '21', '249', '250', '251', '252', '253', '254', '255', '256', '257', '258', '259', '260', '261', '262', '263', '264', '265', '266', '267', '268', '269', '270', '271', '272', '273', '274', '275', '277', '278', '279', '3', '4', '5', '559', '560', '561', '562', '563', '564', '565', '566', '567', '568', '569', '570', '571', '572', '573', '574', '575', '576', '577', '578', '579', '580', '582', '583', '586', '587', '588', '589', '6', '7', '9'},
2: { '14', '17', '19', '20', '22', '23', '24', '25', '26', '28', '29', '30', '31', '585', '8'},
3: {'276', '581', '584', '27'},
4: {'15'}}

LDA gesture-gesture similarity matrix results:

{ 1: { '10', '11', '12', '14', '16', '19', '2', '249', '250', '252', '253', '254', '255', '256', '257', '258', '259', '260', '261', '262', '263', '264', '265', '266', '268', '270', '271', '273', '274', '275', '276', '278', '279', '3', '31', '4', '5', '559', '560', '561', '563', '566', '568', '570', '571', '572', '573', '574', '575', '576', '577', '578', '580', '581', '582', '583', '584', '585', '586', '587', '588', '589', '6', '8', '9'},
2: { '1', '13', '17', '18', '20', '21', '22', '23', '24', '25', '26', '267', '269', '27', '272', '277', '28', '29', '30', '562', '564', '565', '567', '579'},
3: {'7'},
4: {'569', '251', '15'}}

Edit distance gesture-gesture similarity matrix results:

{ 1: { '1', '12', '14', '19', '253', '254', '263', '266', '267', '270', '271', '273', '274', '275', '276', '278', '279', '4', '570', '577', '578', '580', '581', '584', '585', '587', '588', '589', '8'},
2: { '15', '17', '20', '22', '23', '24', '25', '255', '259', '26', '260', '27', '272', '28', '29', '30', '31', '564', '565', '568', '569', '571'},
3: { '11', '13', '16', '2', '249', '250', '251', '256', '257', '258', '261', '265', '5', '559', '561', '576', '579', '586', '7', '9'},

4: { '10', '18', '21', '252', '262', '264', '268', '269', '277', '3',
'560', '562', '563', '566', '567', '572', '573', '574', '575', '582',
'583', '6'}}

DTW gesture-gesture similarity matrix results:

{ 1: { '252', '256', '265', '270', '273', '274', '275', '278', '279', '4',
'564', '574', '576', '581'},
2: { '10', '13', '15', '21', '25', '251', '257', '259', '269', '30',
'559', '560', '561', '563', '565', '566', '567', '568', '569', '570',
'571', '579', '6'},
3: { '14', '17', '19', '20', '22', '23', '24', '250', '253', '254', '255',
'258', '26', '260', '263', '264', '266', '267', '27', '271', '28',
'29', '31', '577', '8'},
4: { '1', '11', '12', '16', '18', '2', '249', '261', '262', '268', '272',
'276', '277', '3', '5', '562', '572', '573', '575', '578', '580',
'582', '583', '584', '585', '586', '587', '588', '589', '7', '9'}}

Taking maximum value for finding maximum contribution of a gesture to an Latent Feature

Dot product gesture-gesture similarity matrix results:

{ 1: { '1', '10', '11', '12', '13', '14', '16', '18', '19', '2', '21',
'249', '25', '250', '251', '252', '253', '254', '255', '256', '257',
'258', '259', '26', '260', '261', '262', '263', '264', '265', '266',
'267', '268', '269', '270', '271', '272', '273', '274', '275', '277',
'278', '279', '29', '3', '30', '31', '4', '5', '559', '560', '561',
'562', '563', '564', '565', '566', '567', '568', '569', '570', '571',
'572', '573', '574', '575', '576', '577', '578', '579', '580', '581',
'582', '583', '585', '586', '587', '588', '589', '6', '7', '8',
'9'},
2: {'15'},
3: {'276', '27', '584'},
4: {'28', '20', '22', '24', '17', '23'}}

PCA gesture-gesture similarity matrix results:

{ 1: { '11', '18', '2', '21', '249', '250', '251', '252', '253', '254',
'255', '256', '257', '258', '259', '260', '261', '262', '263', '264',
'265', '266', '267', '268', '269', '27', '271', '273', '276', '277',
'279', '3', '5', '559', '561', '562', '563', '564', '565', '567',
'572', '574', '575', '576', '579', '582', '584', '586', '588', '6',
'7', '9'},
2: { '10', '12', '13', '14', '19', '20', '22', '24', '26', '270', '29',
'30', '31', '566', '568', '569', '570', '571', '573', '578', '585',
'8'},

3: {'560', '577', '587', '580', '1', '28', '25', '274', '272', '278', '23'},
4: {'275', '589', '4', '15', '16', '583', '17', '581'}}

SVD gesture-gesture similarity matrix results:

{ 1: { '1', '10', '11', '12', '13', '16', '18', '2', '21', '249', '250',
'251', '252', '253', '254', '255', '256', '257', '258', '259', '260',
'261', '262', '263', '264', '265', '266', '267', '268', '269', '270',
'271', '272', '273', '274', '275', '277', '278', '279', '3', '4',
'5', '559', '560', '561', '562', '563', '564', '565', '566', '567',
'568', '569', '570', '571', '572', '573', '574', '575', '576', '577',
'578', '579', '580', '582', '583', '586', '587', '588', '589', '6',
'7', '9'},
2: { '14', '17', '19', '20', '22', '23', '24', '25', '26', '28', '29',
'30', '31', '585', '8'},
3: {'276', '27', '581', '584'},
4: {'15'}}

NMF gesture-gesture similarity matrix results:

{ 1: { '1', '10', '11', '12', '13', '16', '18', '2', '21', '249', '250',
'251', '252', '253', '254', '255', '256', '257', '258', '259', '260',
'261', '262', '263', '264', '265', '266', '267', '268', '269', '270',
'271', '272', '273', '274', '275', '277', '278', '279', '3', '4',
'5', '559', '560', '561', '562', '563', '564', '565', '566', '567',
'568', '569', '570', '571', '572', '573', '574', '575', '576', '577',
'578', '579', '580', '582', '583', '586', '587', '588', '589', '6',
'7', '9'},
2: { '14', '17', '19', '20', '22', '23', '24', '25', '26', '28', '29',
'30', '31', '585', '8'},
3: {'276', '27', '581', '584'},
4: {'15'}}

LDA gesture-gesture similarity matrix results:

{ 1: { '10', '11', '12', '13', '14', '16', '18', '19', '2', '21', '249',
'250', '252', '253', '254', '255', '256', '257', '258', '259', '260',
'261', '262', '263', '264', '265', '266', '268', '270', '271', '273',
'274', '275', '276', '277', '278', '279', '3', '31', '4', '5', '559',
'560', '561', '563', '564', '565', '566', '567', '568', '570', '571',
'572', '573', '574', '575', '576', '577', '578', '579', '580', '581',
'582', '583', '584', '585', '586', '587', '588', '589', '6', '8',
'9'},
2: { '1', '17', '20', '22', '23', '24', '25', '26', '27', '272', '28',
'29', '30'},
3: {'251', '269', '569', '267', '562', '7'},
4: {'15'}}

Edit distance gesture-gesture similarity matrix results:

{ 1: { '1', '11', '12', '14', '19', '2', '253', '254', '255', '256', '258',
'259', '260', '262', '263', '264', '265', '266', '267', '268', '270',
'271', '273', '274', '275', '276', '277', '278', '279', '3', '4',
'559', '564', '565', '568', '569', '570', '571', '576', '577', '578',
'580', '581', '584', '585', '586', '587', '588', '589', '6', '8'},
2: { '15', '17', '20', '22', '23', '24', '25', '26', '27', '272', '28',
'29', '30', '31'},
3: { '13', '16', '18', '21', '249', '250', '251', '257', '261', '269',
'5', '561', '579', '7', '9'},
4: { '10', '252', '560', '562', '563', '566', '567', '572', '573', '574',
'575', '582', '583'}}

DTW gesture-gesture similarity matrix results:

{ 1: { '1', '11', '12', '14', '15', '17', '19', '2', '20', '22', '23', '24',
'25', '252', '256', '26', '262', '265', '268', '27', '270', '272',
'273', '274', '275', '276', '277', '278', '279', '28', '29', '30',
'31', '4', '559', '564', '570', '574', '576', '581', '8', '9'},
2: { '10', '13', '16', '18', '21', '249', '251', '257', '259', '269', '3',
'5', '560', '561', '563', '565', '566', '567', '568', '569', '571',
'579', '7'},
3: { '250', '253', '254', '255', '258', '260', '261', '263', '264', '266',
'267', '271', '577', '6'},
4: { '562', '572', '573', '575', '578', '580', '582', '583', '584', '585',
'586', '587', '588', '589'}}

Task 4b:

Taking absolute value for finding maximum contribution of a gesture to an Latent Feature
Dot product gesture-gesture similarity matrix results:

{ 1: {'276', '584', '27'},
2: {'26', '22', '20', '23', '17', '24', '28'},
3: { '1', '10', '11', '12', '13', '14', '16', '18', '19', '2', '21',
'249', '25', '250', '251', '252', '253', '254', '255', '256', '257',
'258', '259', '260', '261', '262', '263', '264', '265', '266', '267',
'268', '269', '270', '271', '272', '273', '274', '275', '277', '278',
'279', '29', '3', '30', '31', '4', '5', '559', '560', '561', '562',
'563', '564', '565', '566', '567', '568', '569', '570', '571', '572',
'573', '574', '575', '576', '577', '578', '579', '580', '581', '582',
'583', '585', '586', '587', '588', '589', '6', '7', '8', '9'},
4: {'15'}}

PCA gesture-gesture similarity matrix results:

{ 1: { '10', '11', '12', '13', '16', '18', '21', '249', '250', '251', '252', '253', '254', '256', '257', '258', '259', '260', '261', '263', '264', '265', '266', '267', '268', '269', '271', '273', '275', '279', '3', '5', '559', '561', '562', '563', '564', '565', '567', '568', '569', '571', '572', '575', '576', '579', '582', '586', '588', '589', '6', '7', '9'},
2: { '14', '17', '20', '22', '23', '26', '270', '28', '29', '31', '566', '570', '573', '578', '587', '8'},
3: { '1', '2', '255', '262', '272', '274', '277', '278', '4', '560', '574', '577', '580'},
4: {'25', '276', '581', '30', '584', '19', '583', '585', '24', '15', '27'}}

SVD gesture-gesture similarity matrix results:

{ 1: { '1', '12', '15', '21', '252', '253', '255', '258', '262', '263', '271', '273', '274', '277', '278', '4', '5', '562', '563', '577', '579', '580', '581', '583', '587', '588'},
2: {'276', '584', '27'},
3: { '10', '11', '13', '16', '18', '2', '249', '250', '251', '254', '256', '257', '259', '260', '261', '264', '265', '266', '267', '268', '269', '270', '272', '275', '279', '3', '30', '559', '560', '561', '564', '565', '566', '567', '568', '569', '570', '571', '572', '573', '574', '575', '576', '578', '582', '586', '589', '6', '7', '8', '9'},
4: { '14', '17', '19', '20', '22', '23', '24', '25', '26', '28', '29', '31', '585'}}

NMF gesture-gesture similarity matrix results:

{ 1: { '1', '10', '11', '12', '19', '21', '25', '252', '253', '255', '258', '262', '263', '266', '268', '271', '273', '274', '277', '278', '279', '30', '4', '5', '561', '562', '563', '567', '571', '573', '577', '579', '580', '581', '583', '585', '586', '587', '588'},
2: {'276', '584', '15', '27'},
3: { '13', '16', '18', '2', '249', '250', '251', '254', '256', '257', '259', '260', '261', '264', '265', '267', '269', '270', '272', '275', '3', '31', '559', '560', '564', '565', '566', '568', '569', '570', '572', '574', '575', '576', '578', '582', '589', '6', '7', '8', '9'},
4: {'26', '29', '22', '20', '23', '14', '17', '24', '28'}}

LDA gesture-gesture similarity matrix results:

{ 1: { '10', '11', '12', '14', '15', '252', '255', '256', '257', '258', '26', '261', '263', '265', '266', '267', '268', '269', '27', '270',

'271', '272', '273', '274', '278', '4', '5', '562', '563', '567',
'568', '571', '577', '579', '581', '583', '585', '587', '588', '6',
'8'},
2: { '16', '17', '18', '2', '20', '23', '24', '249', '251', '260', '264',
'275', '276', '3', '30', '559', '566', '569', '575', '576', '589',
'7'},
3: {'584', '254', '22', '25', '570', '29', '582', '580'},
4: { '1', '13', '19', '21', '250', '253', '259', '262', '277', '279',
'28', '31', '560', '561', '564', '565', '572', '573', '574', '578',
'586', '9'}}

Edit distance gesture-gesture similarity matrix results:

{ 1: { '14', '15', '17', '20', '22', '23', '24', '25', '26', '27', '272',
'28', '29', '30', '31'},
2: { '10', '560', '562', '563', '564', '566', '567', '568', '571', '572',
'573', '575', '578', '580', '581', '582', '583', '584', '585', '587',
'588', '589', '8'},
3: { '1', '11', '12', '19', '2', '253', '254', '256', '258', '262', '263',
'265', '266', '267', '268', '270', '271', '273', '274', '275', '276',
'277', '278', '279', '4', '559', '570', '574', '576', '577', '586',
'6'},
4: { '13', '16', '18', '21', '249', '250', '251', '252', '255', '257',
'259', '260', '261', '264', '269', '3', '5', '561', '565', '569',
'579', '7', '9'}}

DTW gesture-gesture similarity matrix results:

{ 1: { '562', '564', '566', '567', '572', '575', '576', '578', '580', '582',
'583', '584', '585', '586', '587', '588', '589'},
2: { '10', '13', '16', '18', '21', '249', '251', '252', '257', '259',
'269', '3', '5', '560', '561', '563', '565', '568', '569', '571',
'579', '581', '7', '9'},
3: { '2', '250', '253', '254', '255', '256', '258', '260', '261', '262',
'263', '264', '265', '266', '267', '271', '272', '273', '274', '275',
'277', '278', '4', '559', '570', '573', '574', '577', '6'},
4: { '1', '11', '12', '14', '15', '17', '19', '20', '22', '23', '24',
'25', '26', '268', '27', '270', '276', '279', '28', '29', '30', '31',
'8'}}

Not taking abs

Dot product gesture-gesture similarity matrix results:

{ 1: {'276', '27', '584'},
2: {'26', '28', '20', '22', '24', '17', '23'},
3: { '1', '10', '11', '12', '13', '14', '16', '18', '19', '2', '21',
'249', '25', '250', '251', '252', '253', '254', '255', '256', '257',

'258', '259', '260', '261', '262', '263', '264', '265', '266', '267',
'268', '269', '270', '271', '272', '273', '274', '275', '277', '278',
'279', '29', '3', '30', '31', '4', '5', '559', '560', '561', '562',
'563', '564', '565', '566', '567', '568', '569', '570', '571', '572',
'573', '574', '575', '576', '577', '578', '579', '580', '581', '582',
'583', '585', '586', '587', '588', '589', '6', '7', '8', '9'},
4: {'15'}}

PCA gesture-gesture similarity matrix results:

{ 1: { '10', '11', '12', '13', '16', '18', '21', '249', '250', '251', '252',
'253', '254', '256', '257', '258', '259', '260', '261', '263', '264',
'265', '266', '267', '268', '269', '271', '273', '275', '279', '3',
'5', '559', '561', '562', '563', '564', '565', '567', '568', '569',
'571', '572', '575', '576', '579', '582', '586', '588', '589', '6',
'7', '9'},
2: { '14', '17', '20', '22', '23', '26', '270', '28', '29', '31', '566',
'570', '573', '578', '587', '8'},
3: { '1', '2', '255', '262', '272', '274', '277', '278', '4', '560',
'574', '577', '580'},
4: {'27', '19', '15', '583', '276', '30', '25', '24', '585', '581', '584'}}

SVD gesture-gesture similarity matrix results:

{ 1: { '1', '12', '15', '21', '252', '253', '255', '258', '262', '263',
'271', '273', '274', '277', '278', '4', '5', '562', '563', '577',
'579', '580', '581', '583', '587', '588'},
2: {'276', '27', '584'},
3: { '10', '11', '13', '16', '18', '2', '249', '250', '251', '254', '256',
'257', '259', '260', '261', '264', '265', '266', '267', '268', '269',
'270', '272', '275', '279', '3', '30', '559', '560', '561', '564',
'565', '566', '567', '568', '569', '570', '571', '572', '573', '574',
'575', '576', '578', '582', '586', '589', '6', '7', '8', '9'},
4: { '14', '17', '19', '20', '22', '23', '24', '25', '26', '28', '29',
'31', '585'}}

NMF gesture-gesture similarity matrix results:

{ 1: { '1', '10', '11', '12', '19', '21', '25', '252', '253', '255', '258',
'262', '263', '266', '268', '271', '273', '274', '277', '278', '279',
'30', '4', '5', '561', '562', '563', '567', '571', '573', '577',
'579', '580', '581', '583', '585', '586', '587', '588'},
2: {'27', '15', '276', '584'},
3: { '13', '16', '18', '2', '249', '250', '251', '254', '256', '257',
'259', '260', '261', '264', '265', '267', '269', '270', '272', '275',
'3', '31', '559', '560', '564', '565', '566', '568', '569', '570',
'572', '574', '575', '576', '578', '582', '589', '6', '7', '8',

'9'},
4: {'26', '28', '20', '22', '14', '24', '29', '17', '23'}}

LDA gesture-gesture similarity matrix results:

{ 1: { '10', '11', '12', '14', '15', '252', '255', '256', '257', '258', '26', '261', '263', '265', '266', '267', '268', '269', '27', '270', '271', '272', '273', '274', '278', '4', '5', '562', '563', '567', '568', '571', '577', '579', '581', '583', '585', '587', '588', '6', '8'},
2: { '16', '17', '18', '2', '20', '23', '24', '249', '251', '260', '264', '275', '276', '3', '30', '559', '566', '569', '575', '576', '589', '7'},
3: {'22', '584', '580', '254', '582', '29', '25', '570'},
4: { '1', '13', '19', '21', '250', '253', '259', '262', '277', '279', '28', '31', '560', '561', '564', '565', '572', '573', '574', '578', '586', '9'}}

Edit distance gesture-gesture similarity matrix results:

{ 1: { '14', '15', '17', '20', '22', '23', '24', '25', '26', '27', '272', '28', '29', '30', '31'},
2: { '10', '560', '562', '563', '564', '566', '567', '568', '571', '572', '573', '575', '578', '580', '581', '582', '583', '584', '585', '587', '588', '589', '8'},
3: { '1', '11', '12', '19', '2', '253', '254', '256', '258', '262', '263', '265', '266', '267', '268', '270', '271', '273', '274', '275', '276', '277', '278', '279', '4', '559', '570', '574', '576', '577', '586', '6'},
4: { '13', '16', '18', '21', '249', '250', '251', '252', '255', '257', '259', '260', '261', '264', '269', '3', '5', '561', '565', '569', '579', '7', '9'}}

DTW gesture-gesture similarity matrix results:

{ 1: { '562', '564', '566', '567', '572', '575', '576', '578', '580', '582', '583', '584', '585', '586', '587', '588', '589'},
2: { '10', '13', '16', '18', '21', '249', '251', '252', '257', '259', '269', '3', '5', '560', '561', '563', '565', '568', '569', '571', '579', '581', '7', '9'},
3: { '2', '250', '253', '254', '255', '256', '258', '260', '261', '262', '263', '264', '265', '266', '267', '271', '272', '273', '274', '275', '277', '278', '4', '559', '570', '573', '574', '577', '6'},
4: { '1', '11', '12', '14', '15', '17', '19', '20', '22', '23', '24', '25', '26', '268', '27', '270', '276', '279', '28', '29', '30', '31', '8'}}

Task 4c:

In this task, the proposed methodology is to use the gesture-gesture similarity matrix obtained in task 3 from the different user options and cluster the gestures into p groups using the KMeans clustering algorithm (where k = p).

KMeans Algorithm:

Step 0: The input to the KMeans algorithm is a gesture gesture similarity matrix. Dimensions = n x n where n = number of gesture objects. This gesture gesture similarity matrix obtained from Task 3 represents each gesture object in terms of all the other gestures in the dataset.

Step 1: Initialization step includes randomly choosing k samples (gesture objects) from the dataset as initial centroids of k clusters. This is because initially we do not know where the center of each cluster lies. A centroid is the center of a cluster.

Step 2: Cluster assignment step consists of assigning all the data samples (gesture objects) to the closest (most similar) centroid from the above step. The distance function we have used to find the closest centroid is Euclidean distance. Thus, k initial clusters will be formed.

Step 3: This step includes iteratively improving the centroids. From the clusters formed in the above step, we recompute the new centroid of each cluster by taking the mean value of all the data samples in a cluster. We keep iterating this step until the centroids stop changing i.e. the cluster assignments become constant. We say that the KMeans algorithm has converged.

The results or final cluster assignments of KMeans majorly depend on the initial random assignment of centroids and it can get stuck on a local solution. This is why we implemented the algorithm with multiple random starts and chose the cluster assignment with the minimum average distance of all points from the centroids of their cluster assignment. We also used python's Sklearn KMeans implementation to compare our results:

```
kmeans = KMeans(n_clusters=p, n_init=random_starts, max_iter=max_iterations, verbose=0, random_state=2).fit(sim_mat)
```

The following **KMeans clustering results** are shown for **Model = TF-IDF**, **n_random_starts = 100, p = 4,** and **max_iterations = 100** and different user options for both our implementation and python's sklearn KMeans implementation:

a. **Dot product** gesture-gesture similarity matrix - KMeans clustering results:

   K Means Output:
   min_avg_dist: 0.00040056879908992484 , n_random_starts: 100

   Cluster: 1 ; n_gestures: 1 ;
   Gestures: ['27']

   Cluster: 2 ; n_gestures: 90 ;

Gestures: ['1', '10', '11', '12', '13', '14', '16', '18', '19', '2', '20', '21', '22', '23', '24', '249', '25', '250', '251', '252', '253', '254', '255', '256', '257', '258', '259', '26', '260', '261', '262', '263', '264', '265', '266', '267', '268', '269', '270', '271', '272', '273', '274', '275', '276', '277', '278', '279', '28', '29', '3', '30', '31', '4', '5', '559', '560', '561', '562', '563', '564', '565', '566', '567', '568', '569', '570', '571', '572', '573', '574', '575', '576', '577', '578', '579', '580', '581', '582', '583', '584', '585', '586', '587', '588', '589', '6', '7', '8', '9']

Cluster: 3 ; n_gestures: 1 ;
Gestures: ['17']

Cluster: 4 ; n_gestures: 1 ;
Gestures: ['15']

Sklearn KMeans Output:

Cluster: 1 ; n_gestures: 1 ;
Gestures: ['15']

Cluster: 2 ; n_gestures: 1 ;
Gestures: ['27']

Cluster: 3 ; n_gestures: 90 ;
Gestures: ['1', '10', '11', '12', '13', '14', '16', '18', '19', '2', '20', '21', '22', '23', '24', '249', '25', '250', '251', '252', '253', '254', '255', '256', '257', '258', '259', '26', '260', '261', '262', '263', '264', '265', '266', '267', '268', '269', '270', '271', '272', '273', '274', '275', '276', '277', '278', '279', '28', '29', '3', '30', '31', '4', '5', '559', '560', '561', '562', '563', '564', '565', '566', '567', '568', '569', '570', '571', '572', '573', '574', '575', '576', '577', '578', '579', '580', '581', '582', '583', '584', '585', '586', '587', '588', '589', '6', '7', '8', '9']

Cluster: 4 ; n_gestures: 1 ;
Gestures: ['17']

As we can see above, while using dot-product similarity matrix, most of the gesture objects get assigned to 1 cluster for our implementation as well as sklearn implementation with random_start value 100 and max_iteration value 100. We can infer that dot product similarity is not a very good similarity measure to differentiate among the 3 class data.

b. **PCA** gesture-gesture similarity matrix KMeans clustering results:

K Means Output:
min_avg_dist: 1.8137200020337847 , n_random_starts: 100

Cluster: 1 ; n_gestures: 6 ;
Gestures: ['15', '27', '276', '581', '583', '584']

Cluster: 2 ; n_gestures: 67 ;
Gestures: ['1', '10', '11', '12', '13', '16', '18', '2', '21', '249', '250', '251', '252', '253', '254', '255', '256', '257', '258', '259', '260', '261', '262', '263', '264', '265', '266', '267', '268', '269', '271', '273', '274', '275', '277', '278', '279', '3', '4', '5', '559', '560', '561', '562', '563', '564', '565', '566', '567', '568', '569', '571', '572', '573', '574', '575', '576', '577', '579', '580', '582', '586', '588', '589', '6', '7', '9']

Cluster: 3 ; n_gestures: 10 ;
Gestures: ['14', '570', '17', '270', '272', '578', '23', '587', '31', '8']

Cluster: 4 ; n_gestures: 10 ;
Gestures: ['19', '20', '22', '24', '25', '28', '29', '585', '30', '26']

Sklearn K Means Output:

Cluster: 1 ; n_gestures: 16 ;
Gestures: ['14', '570', '17', '19', '20', '22', '23', '24', '25', '28', '29', '585', '30', '31', '8', '26']

Cluster: 2 ; n_gestures: 59 ;
Gestures: ['10', '11', '13', '16', '18', '2', '21', '249', '250', '251', '252', '253', '254', '255', '256', '257', '258', '259', '260', '261', '262', '263', '264', '265', '266', '267', '268', '269', '271', '273', '274', '275', '277', '279', '3', '4', '5', '559', '560', '561', '562', '563', '564', '565', '567', '569', '571', '572', '574', '575', '576', '579', '582', '586', '588', '589', '6', '7', '9']

Cluster: 3 ; n_gestures: 6 ;
Gestures: ['15', '27', '276', '581', '583', '584']

Cluster: 4 ; n_gestures: 12 ;
Gestures: ['1', '566', '12', '568', '573', '270', '272', '577', '578', '580', '278', '587']

We can observe that PCA is considerably better than dot product in differentiating between the objects. Also, our implementation and Sklearn's K Means implementation shows some overlap in the cluster assignments.But it still cannot differentiate between the 3 class gestures.

c. **SVD** gesture-gesture similarity matrix KMeans clustering results:

K Means Output:
min_avg_dist: 0.6828396213565449 , n_random_starts: 100

Cluster: 1 ; n_gestures: 10 ;
Gestures: ['15', '17', '27', '20', '22', '23', '24', '276', '28', '584']

Cluster: 2 ; n_gestures: 18 ;

Gestures: ['262', '263', '265', '266', '271', '273', '277', '249', '279', '250', '252', '253', '254', '256', '257', '258', '260', '261']

Cluster: 3 ; n_gestures: 53 ;
Gestures: ['1', '10', '11', '12', '13', '16', '18', '2', '21', '251', '255', '259', '264', '267', '268', '269', '270', '272', '274', '275', '278', '3', '4', '5', '559', '560', '561', '562', '563', '564', '565', '566', '567', '568', '569', '571', '572', '573', '574', '575', '576', '577', '578', '579', '580', '582', '586', '587', '588', '589', '6', '7', '9']

Cluster: 4 ; n_gestures: 12 ;
Gestures: ['14', '570', '19', '581', '25', '583', '29', '585', '30', '31', '8', '26']

Sklearn K Means Output:

Cluster: 1 ; n_gestures: 68 ;
Gestures: ['1', '10', '11', '12', '13', '16', '18', '2', '21', '249', '250', '251', '252', '253', '254', '255', '256', '257', '258', '259', '260', '261', '262', '263', '264', '265', '266', '267', '268', '269', '271', '272', '273', '274', '275', '277', '278', '279', '3', '4', '5', '559', '560', '561', '562', '563', '564', '565', '566', '567', '568', '569', '571', '572', '573', '574', '575', '576', '577', '579', '580', '582', '586', '588', '589', '6', '7', '9']

Cluster: 2 ; n_gestures: 7 ;
Gestures: ['20', '22', '276', '24', '23', '28', '584']

Cluster: 3 ; n_gestures: 3 ;
Gestures: ['17', '27', '15']

Cluster: 4 ; n_gestures: 15 ;
Gestures: ['14', '570', '270', '19', '578', '581', '25', '583', '29', '585', '30', '31', '587', '8', '26']

We can observe that the gesture-gesture similarity matrix created through SVD is able to cluster most of the gestures starting with 2 (which we know belong to the same class data) in a single cluster 2. It assigns all but 4 of the gestures starting with 5 to the same cluster.

d. **NMF** gesture-gesture similarity matrix KMeans clustering results:

K Means Output:
min_avg_dist: 0.7067497915168882 , n_random_starts: 100

Cluster: 1 ; n_gestures: 10 ;
Gestures: ['15', '17', '27', '20', '22', '23', '24', '276', '28', '584']

Cluster: 2 ; n_gestures: 9 ;
Gestures: ['14', '570', '19', '581', '25', '585', '30', '31', '8']

Cluster: 3 ; n_gestures: 2 ;
Gestures: ['29', '26']

Cluster: 4 ; n_gestures: 72 ;
Gestures: ['1', '10', '11', '12', '13', '16', '18', '2', '21', '249', '250', '251', '252', '253', '254', '255', '256', '257', '258', '259', '260', '261', '262', '263', '264', '265', '266', '267', '268', '269', '270', '271', '272', '273', '274', '275', '277', '278', '279', '3', '4', '5', '559', '560', '561', '562', '563', '564', '565', '566', '567', '568', '569', '571', '572', '573', '574', '575', '576', '577', '578', '579', '580', '582', '583', '586', '587', '588', '589', '6', '7', '9']

Sklearn K Means Output:

Cluster: 1 ; n_gestures: 72 ;
Gestures: ['1', '10', '11', '12', '13', '16', '18', '2', '21', '249', '250', '251', '252', '253', '254', '255', '256', '257', '258', '259', '260', '261', '262', '263', '264', '265', '266', '267', '268', '269', '270', '271', '272', '273', '274', '275', '277', '278', '279', '3', '4', '5', '559', '560', '561', '562', '563', '564', '565', '566', '567', '568', '569', '571', '572', '573', '574', '575', '576', '577', '578', '579', '580', '582', '583', '586', '587', '588', '589', '6', '7', '9']

Cluster: 2 ; n_gestures: 6 ;
Gestures: ['17', '20', '22', '23', '24', '28']

Cluster: 3 ; n_gestures: 11 ;
Gestures: ['14', '570', '19', '581', '25', '29', '585', '30', '31', '8', '26']

Cluster: 4 ; n_gestures: 4 ;
Gestures: ['27', '584', '15', '276']

NMF too is unable to differentiate the 3 class data correctly, although all but 3 of the gestures belonging to class 5 were assigned to the same cluster. There is some overlap between our implementation and Sklearn' s results.

e. LDA gesture-gesture similarity matrix KMeans clustering results:

K Means Output:
min_avg_dist: 0.012963109858768206 , n_random_starts: 100

Cluster: 1 ; n_gestures: 41 ;
Gestures: ['10', '13', '16', '18', '21', '249', '250', '251', '252', '253', '254', '255', '256', '257', '260', '261', '263', '266', '267', '268', '269', '273', '275', '276', '277', '3', '5', '560', '561', '562', '563', '564', '565', '566', '567', '568', '569', '571', '575', '579', '7']

Cluster: 2 ; n_gestures: 41 ;

Gestures: ['1', '11', '12', '14', '19', '2', '258', '259', '26', '262', '264', '265', '270', '271', '274', '278', '279', '29', '31', '4', '559', '570', '572', '573', '574', '576', '577', '578', '580', '581', '582', '583', '584', '585', '586', '587', '588', '589', '6', '8', '9']

Cluster: 3 ; n_gestures: 10 ;
Gestures: ['17', '27', '272', '20', '22', '23', '24', '25', '28', '30']

Cluster: 4 ; n_gestures: 1 ;
Gestures: ['15']

Sklearn K Means Output:

Cluster: 1 ; n_gestures: 10 ;
Gestures: ['17', '27', '272', '20', '22', '23', '24', '25', '28', '30']

Cluster: 2 ; n_gestures: 42 ;
Gestures: ['1', '11', '12', '14', '19', '2', '258', '259', '26', '262', '264', '265', '270', '271', '274', '276', '278', '279', '29', '31', '4', '559', '570', '572', '573', '574', '576', '577', '578', '580', '581', '582', '583', '584', '585', '586', '587', '588', '589', '6', '8', '9']

Cluster: 3 ; n_gestures: 40 ;
Gestures: ['10', '13', '16', '18', '21', '249', '250', '251', '252', '253', '254', '255', '256', '257', '260', '261', '263', '266', '267', '268', '269', '273', '275', '277', '3', '5', '560', '561', '562', '563', '564', '565', '566', '567', '568', '569', '571', '575', '579', '7']

Cluster: 4 ; n_gestures: 1 ;
Gestures: ['15']

We can see that the KMeans cluster assignments using the LDA gesture-gesture similarity matrix are almost exactly the same as that of Sklearn's KMeans cluster assignments. The fact that they both converged to a similar assignment within 100 random starts, might convey that this cluster assignment is a global minima of average distance of every point from the centroid of its cluster.

f. Edit distance gesture-gesture similarity matrix KMeans clustering results:

K Means Output:
min_avg_dist: 0.7013879618787716 , n_random_starts: 100

Cluster: 1 ; n_gestures: 1 ;
Gestures: ['579']

Cluster: 2 ; n_gestures: 48 ;

Gestures: ['10', '13', '16', '18', '21', '249', '250', '251', '252', '253', '254', '255', '256', '257', '258', '259', '260', '261', '263', '264', '266', '267', '269', '271', '275', '276', '277', '3', '5', '560', '561', '562', '563', '564', '565', '567', '568', '569', '571', '572', '573', '574', '578', '580', '588', '589', '7', '9']

Cluster: 3 ; n_gestures: 34 ;
Gestures: ['1', '11', '12', '14', '19', '2', '23', '24', '262', '265', '268', '270', '273', '274', '278', '279', '29', '31', '4', '559', '566', '570', '575', '576', '577', '581', '582', '583', '584', '585', '586', '587', '6', '8']

Cluster: 4 ; n_gestures: 10 ;
Gestures: ['15', '17', '27', '272', '20', '22', '25', '28', '30', '26']

Sklearn K Means Output:

Cluster: 1 ; n_gestures: 12 ;
Gestures: ['15', '17', '27', '272', '20', '22', '23', '24', '25', '28', '30', '26']

Cluster: 2 ; n_gestures: 35 ;
Gestures: ['10', '250', '252', '253', '255', '259', '260', '263', '264', '266', '267', '271', '275', '276', '277', '279', '3', '560', '562', '563', '564', '565', '567', '568', '569', '571', '572', '574', '578', '580', '582', '583', '588', '589', '9']

Cluster: 3 ; n_gestures: 33 ;
Gestures: ['1', '11', '12', '14', '19', '2', '254', '256', '258', '262', '265', '268', '270', '273', '274', '278', '29', '31', '4', '559', '566', '570', '573', '575', '576', '577', '581', '584', '585', '586', '587', '6', '8']

Cluster: 4 ; n_gestures: 13 ;
Gestures: ['13', '7', '16', '269', '18', '21', '579', '249', '251', '5', '257', '561', '261']

There is some overlap between our implementation and sklearn's implementation, but the classes are not differentiated even by using the edit distance gesture-gesture similarity matrix.

g. DTW gesture-gesture similarity matrix KMeans clustering results:

K Means Output:
min_avg_dist: 0.7506914334297856 , n_random_starts: 100

Cluster: 1 ; n_gestures: 56 ;
Gestures: ['1', '11', '12', '14', '15', '17', '19', '2', '20', '22', '23', '24', '25', '250', '253', '254', '255', '256', '258', '26', '260', '262', '263', '264', '265', '266', '267', '268', '27', '270', '271', '272', '273', '274', '275', '276', '277', '278', '279', '28', '29', '30', '31', '4', '559', '570', '572', '573', '574', '575', '577', '584', '585', '586', '587', '6']

Cluster: 2 ; n_gestures: 23 ;
Gestures: ['10', '251', '252', '259', '560', '562', '563', '564', '565', '566', '567', '568', '569', '571', '576', '578', '580', '581', '582', '583', '588', '589', '8']

Cluster: 3 ; n_gestures: 2 ;
Gestures: ['21', '579']

Cluster: 4 ; n_gestures: 12 ;
Gestures: ['13', '7', '16', '269', '18', '9', '249', '3', '5', '257', '561', '261']

Sklearn K Means Output:
Cluster: 1 ; n_gestures: 22 ;
Gestures: ['10', '560', '562', '563', '564', '566', '567', '568', '569', '571', '576', '578', '580', '581', '582', '583', '584', '585', '586', '587', '588', '589']

Cluster: 2 ; n_gestures: 17 ;
Gestures: ['14', '15', '17', '27', '19', '270', '20', '22', '23', '24', '25', '28', '29', '30', '31', '8', '26']

Cluster: 3 ; n_gestures: 17 ;
Gestures: ['565', '13', '7', '16', '269', '18', '21', '561', '579', '9', '249', '251', '3', '5', '257', '259', '261']

Cluster: 4 ; n_gestures: 37 ;
Gestures: ['1', '11', '12', '2', '250', '252', '253', '254', '255', '256', '258', '260', '262', '263', '264', '265', '266', '267', '268', '271', '272', '273', '274', '275', '276', '277', '278', '279', '4', '559', '570', '572', '573', '574', '575', '577', '6']

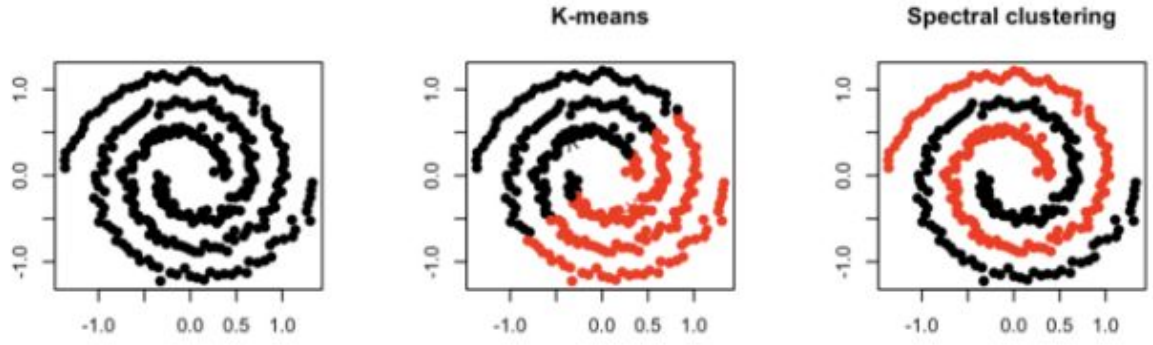
DTW performed considerably well in differentiating and grouping the class 5 gestures in a single cluster as compared to the previously used methods. Sklearn's implementation was also able to distinguish the class 0 data, which means that it found a better minima than our implementation due to a different random start.

In conclusion of task 3C, we can say that although the results of our implementation of KMeans were comparable to those of Sklearn, it was not able to discriminate between the 3 class data extremely well. This can be attributed to the fact that KMeans does not always converge to the global minima, instead is greatly dependent on the initial random assignment. Even though we tried to overcome this by accommodating multiple random starts in our implementation, it does not guarantee finding a global solution, since the number of possible initial assignments is quite large. But we did see an improvement in the cluster assignments on increasing the number of random starts as it reduced the minimum average distance of each point from its cluster centroid.

Task 4d:

Clustering methods like k-means rely heavily on the initial random assignment of the centroids. To counter that we made numerous runs using many random initial starts in task 4c. Additionally, techniques like k-means or EM algorithm require convexity on the loss function. Spectral clustering techniques provide us with an approach to clustering that does not rely on shapes of the clusters or on random starts. Spectral clustering takes the connectivity approach to clustering. That is, the points that are immediately next to each other or directly connected to each other are clustered together.

The figure below provides a visual example of the difference between compactness based clustering approaches like k-means and connectivity based approaches like spectral clustering. The first diagram is the representation of the original data. The next diagram shows how k-means goes about clustering it followed by spectral-clustering. We had two disconnected regions in the data which led to two clusters being obtained due to spectral clustering.



Spectral clustering involves calculating the The Laplacian (L). The Laplacian is defined as follows:

L = D - A

Where A is an adjacency matrix and D is a diagonal matrix of degrees.

$$A_{ij} = \begin{cases} w_{ij} & : \text{weight of edge } (i,j) \\ 0 & : \text{if no edge between } i,j \end{cases}$$

And,

$$d_i = \sum_{\{j|(i,j)\in E\}} w_{ij}$$

We can interpret each di as the sum of the corresponding rows in the adjacency matrix. Thus, D - A gives us the Laplacian.

Spectral clustering can be of two types:

1. Unnormalized
2. Normalized

Unnormalized spectral can be used for dimensionality reduction and bi-class classification. Incase of multi-class classification, we need to use Normalized Spectral clustering.
For Normalized Spectral Clustering, we calculate normalized Laplacian as follows:

$$L_{norm} = D^{-1/2} L D^{-1/2}$$

In our setting, we consider the gesture-gesture similarity matrix obtained by any of the approaches as the adjacency matrix A. We then use that to calculate D. In our case, P can be greater than 2 and hence we use Normalized spectral clustering.
Following is Normalized Spectral clustering algorithm by Ng, Jordan and Weiss that we use:

Input: Similarity matrix (n * n) and the number of clusters to construct.

- Let A be the weighted adjacency matrix having n rows and n columns
- Let D be degree matrix
- Compute the unnormalized Laplacian (L = D - A)
- Compute the Normalized Laplacian (L_n) using L and D
- Compute the first k eigenvectors of Ln (u_1, u_2…….u_k)
- Let U be the matrix containing u1….uk as columns, thus U has n rows and k columns
- Form the matrix T from U by normalizing the rows to norm 1
- For i = 1 to n, let y_i be the vector corresponding to the i-th row of matrix T
- Cluster the points (y_i) with k-means algorithm into clusters C_1…..C_k

Output: Clusters A_1, ……, A_k with Ai = { j | $y_j \in C_i$ }

Results:

a. **Dot product** gesture-gesture similarity matrix Spectral clustering results:

Spectral Clustering Output:
min_avg_dist: 0.012734400435891843 , n_random_starts: 100

Cluster: 1 ; n_gestures: 1 ;
Gestures: [251]

Cluster: 2 ; n_gestures: 1 ;
Gestures: [569]

Cluster: 3 ; n_gestures: 1 ;
Gestures: [267]

Cluster: 4 ; n_gestures: 90 ;
Gestures: [1, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 2, 20, 21, 22, 23, 24, 249, 25, 250, 252, 253, 254, 255, 256, 257, 258, 259, 26, 260, 261, 262, 263, 264, 265, 266, 268, 269, 27, 270, 271, 272, 273, 274, 275, 276, 277, 278, 279, 28, 29, 3, 30, 31, 4, 5, 559, 560, 561, 562, 563, 564, 565, 566, 567, 568, 570, 571, 572, 573, 574, 575, 576, 577, 578, 579, 580, 581, 582, 583, 584, 585, 586, 587, 588, 589, 6, 7, 8, 9]

Spectral Clustering Output with Sklearn's KMeans:

Cluster: 1 ; n_gestures: 90 ;
Gestures: [1, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 2, 20, 21, 22, 23, 24, 249, 25, 250, 252, 253, 254, 255, 256, 257, 258, 259, 26, 260, 261, 262, 263, 264, 265, 266, 268, 269, 27, 270, 271, 272, 273, 274, 275, 276, 277, 278, 279, 28, 29, 3, 30, 31, 4, 5, 559, 560, 561, 562, 563, 564, 565, 566, 567, 568, 570, 571, 572, 573, 574, 575, 576, 577, 578, 579, 580, 581, 582, 583, 584, 585, 586, 587, 588, 589, 6, 7, 8, 9]

Cluster: 2 ; n_gestures: 1 ;
Gestures: [251]

Cluster: 3 ; n_gestures: 1 ;
Gestures: [569]

Cluster: 4 ; n_gestures: 1 ;
Gestures: [267]

The spectral clustering output for the dot-product gesture-gesture similarity matrix is exactly the same for both implementations. However, it is still unable to differentiate among the 3 class data.

b. PCA gesture-gesture similarity matrix Spectral clustering results:

Spectral Clustering Output:
min_avg_dist: 0.03412966901959425 , n_random_starts: 100

Cluster: 1 ; n_gestures: 1 ;
Gestures: [577]

Cluster: 2 ; n_gestures: 21 ;
Gestures: [14, 17, 19, 20, 22, 23, 24, 25, 26, 270, 272, 28, 29, 30, 31, 570, 581, 583, 585, 587, 8]

Cluster: 3 ; n_gestures: 1 ;
Gestures: [15]

Cluster: 4 ; n_gestures: 70 ;
Gestures: [1, 10, 11, 12, 13, 16, 18, 2, 21, 249, 250, 251, 252, 253, 254, 255, 256, 257, 258, 259, 260, 261, 262, 263, 264, 265, 266, 267, 268, 269, 27, 271, 273, 274, 275, 276, 277, 278, 279, 3, 4, 5, 559, 560, 561, 562, 563, 564, 565, 566, 567, 568, 569, 571, 572, 573, 574, 575, 576, 578, 579, 580, 582, 584, 586, 588, 589, 6, 7, 9]

Spectral Clustering Output with Sklearn's KMeans:
Cluster: 1 ; n_gestures: 19 ;
Gestures: [14, 17, 19, 20, 22, 23, 24, 25, 26, 270, 272, 28, 29, 30, 31, 570, 585, 587, 8]

Cluster: 2 ; n_gestures: 68 ;
Gestures: [1, 10, 11, 12, 13, 16, 18, 2, 21, 249, 250, 251, 252, 253, 254, 255, 256, 257, 258, 259, 260, 261, 262, 263, 264, 265, 266, 267, 268, 269, 271, 273, 274, 275, 277, 278, 279, 3, 4, 5, 559, 560, 561, 562, 563, 564, 565, 566, 567, 568, 569, 571, 572, 573, 574, 575, 576, 577, 578, 579, 580, 582, 586, 588, 589, 6, 7, 9]

Cluster: 3 ; n_gestures: 1 ;
Gestures: [15]

Cluster: 4 ; n_gestures: 5 ;
Gestures: [27, 276, 581, 583, 584]

c. **SVD** gesture-gesture similarity matrix Spectral clustering results:

Spectral Clustering Output:
min_avg_dist: 0.0016075234250363575 , n_random_starts: 100

Cluster: 1 ; n_gestures: 1 ;
Gestures: [17]

Cluster: 2 ; n_gestures: 90 ;
Gestures: [1, 10, 11, 12, 13, 14, 16, 18, 19, 2, 20, 21, 22, 23, 24, 249, 25, 250, 251, 252, 253, 254, 255, 256, 257, 258, 259, 26, 260, 261, 262, 263, 264, 265, 266, 267, 268, 269, 270, 271, 272, 273, 274, 275, 276, 277, 278, 279, 28, 29, 3, 30, 31, 4, 5, 559, 560, 561, 562, 563, 564, 565, 566, 567, 568, 569, 570, 571, 572, 573, 574, 575, 576, 577, 578, 579, 580, 581, 582, 583, 584, 585, 586, 587, 588, 589, 6, 7, 8, 9]

Cluster: 3 ; n_gestures: 1 ;
Gestures: [15]

Cluster: 4 ; n_gestures: 1 ;
Gestures: [27]

Spectral Clustering Output with Sklearn's KMeans:
Cluster: 1 ; n_gestures: 90 ;
Gestures: [1, 10, 11, 12, 13, 14, 16, 18, 19, 2, 20, 21, 22, 23, 24, 249, 25, 250, 251, 252, 253, 254, 255, 256, 257, 258, 259, 26, 260, 261, 262, 263, 264, 265, 266, 267, 268, 269, 270, 271, 272, 273, 274, 275, 276, 277, 278, 279, 28, 29, 3, 30, 31, 4, 5, 559, 560, 561, 562, 563, 564, 565, 566, 567, 568, 569, 570, 571, 572, 573, 574, 575, 576, 577, 578, 579, 580, 581, 582, 583, 584, 585, 586, 587, 588, 589, 6, 7, 8, 9]

Cluster: 2 ; n_gestures: 1 ;
Gestures: [15]

Cluster: 3 ; n_gestures: 1 ;
Gestures: [27]

Cluster: 4 ; n_gestures: 1 ;
Gestures: [17]

d. **NMF** gesture-gesture similarity matrix Spectral clustering results:

Spectral Clustering Output:
min_avg_dist: 0.0033079055686366576 , n_random_starts: 100

Cluster: 1 ; n_gestures: 3 ;
Gestures: [17, 28, 23]

Cluster: 2 ; n_gestures: 1 ;
Gestures: [27]

Cluster: 3 ; n_gestures: 1 ;
Gestures: [15]

Cluster: 4 ; n_gestures: 88 ;
Gestures: [1, 10, 11, 12, 13, 14, 16, 18, 19, 2, 20, 21, 22, 24, 249, 25, 250, 251, 252, 253, 254, 255, 256, 257, 258, 259, 26, 260, 261, 262, 263, 264, 265, 266, 267, 268, 269, 270, 271, 272, 273, 274, 275, 276, 277, 278, 279, 29, 3, 30, 31, 4, 5, 559, 560, 561, 562, 563, 564, 565, 566, 567, 568, 569, 570, 571, 572, 573, 574, 575, 576, 577, 578, 579, 580, 581, 582, 583, 584, 585, 586, 587, 588, 589, 6, 7, 8, 9]

Spectral Clustering Output with Sklearn's KMeans:

Cluster: 1 ; n_gestures: 3 ;
Gestures: [17, 28, 23]

Cluster: 2 ; n_gestures: 88 ;

Gestures: [1, 10, 11, 12, 13, 14, 16, 18, 19, 2, 20, 21, 22, 24, 249, 25, 250, 251, 252, 253, 254, 255, 256, 257, 258, 259, 26, 260, 261, 262, 263, 264, 265, 266, 267, 268, 269, 270, 271, 272, 273, 274, 275, 276, 277, 278, 279, 29, 3, 30, 31, 4, 5, 559, 560, 561, 562, 563, 564, 565, 566, 567, 568, 569, 570, 571, 572, 573, 574, 575, 576, 577, 578, 579, 580, 581, 582, 583, 584, 585, 586, 587, 588, 589, 6, 7, 8, 9]

Cluster: 3 ; n_gestures: 1 ;
Gestures: [27]

Cluster: 4 ; n_gestures: 1 ;
Gestures: [15]

e. LDA gesture-gesture similarity matrix Spectral clustering results:

Spectral Clustering Output:
min_avg_dist: 0.18624611613026137 , n_random_starts: 100

Cluster: 1 ; n_gestures: 1 ;
Gestures: [23]

Cluster: 2 ; n_gestures: 3 ;
Gestures: [7, 251, 569]

Cluster: 3 ; n_gestures: 2 ;
Gestures: [27, 15]

Cluster: 4 ; n_gestures: 87 ;
Gestures: [1, 10, 11, 12, 13, 14, 16, 17, 18, 19, 2, 20, 21, 22, 24, 249, 25, 250, 252, 253, 254, 255, 256, 257, 258, 259, 26, 260, 261, 262, 263, 264, 265, 266, 267, 268, 269, 270, 271, 272, 273, 274, 275, 276, 277, 278, 279, 28, 29, 3, 30, 31, 4, 5, 559, 560, 561, 562, 563, 564, 565, 566, 567, 568, 570, 571, 572, 573, 574, 575, 576, 577, 578, 579, 580, 581, 582, 583, 584, 585, 586, 587, 588, 589, 6, 8, 9]

Spectral Clustering Output with Sklearn's KMeans:

Cluster: 1 ; n_gestures: 1 ;
Gestures: [27]

Cluster: 2 ; n_gestures: 90 ;
Gestures: [1, 10, 11, 12, 13, 14, 16, 17, 18, 19, 2, 20, 21, 22, 24, 249, 25, 250, 251, 252, 253, 254, 255, 256, 257, 258, 259, 26, 260, 261, 262, 263, 264, 265, 266, 267, 268, 269, 270, 271, 272, 273, 274, 275, 276, 277, 278, 279, 28, 29, 3, 30, 31, 4, 5, 559, 560,

561, 562, 563, 564, 565, 566, 567, 568, 569, 570, 571, 572, 573, 574, 575, 576, 577, 578, 579, 580, 581, 582, 583, 584, 585, 586, 587, 588, 589, 6, 7, 8, 9]

Cluster: 3 ; n_gestures: 1 ;
Gestures: [23]

Cluster: 4 ; n_gestures: 1 ;
Gestures: [15]

f. Edit Distance gesture-gesture similarity matrix Spectral clustering results:

Spectral Clustering Output:
min_avg_dist: 0.03831056893748361 , n_random_starts: 100

Cluster: 1 ; n_gestures: 1 ;
Gestures: [18]

Cluster: 2 ; n_gestures: 89 ;
Gestures: [1, 10, 11, 12, 13, 14, 15, 16, 17, 19, 2, 20, 21, 22, 23, 24, 249, 25, 250, 251, 252, 253, 254, 255, 256, 258, 259, 26, 260, 262, 263, 264, 265, 266, 267, 268, 269, 27, 270, 271, 272, 273, 274, 275, 276, 277, 278, 279, 28, 29, 3, 30, 31, 4, 5, 559, 560, 561, 562, 563, 564, 565, 566, 567, 568, 569, 570, 571, 572, 573, 574, 575, 576, 577, 578, 580, 581, 582, 583, 584, 585, 586, 587, 588, 589, 6, 7, 8, 9]

Cluster: 3 ; n_gestures: 1 ;
Gestures: [579]

Cluster: 4 ; n_gestures: 2 ;
Gestures: [257, 261]

Spectral Clustering Output with Sklearn's KMeans:
Cluster: 1 ; n_gestures: 89 ;
Gestures: [1, 10, 11, 12, 13, 14, 15, 16, 17, 19, 2, 20, 21, 22, 23, 24, 249, 25, 250, 251, 252, 253, 254, 255, 256, 258, 259, 26, 260, 262, 263, 264, 265, 266, 267, 268, 269, 27, 270, 271, 272, 273, 274, 275, 276, 277, 278, 279, 28, 29, 3, 30, 31, 4, 5, 559, 560, 561, 562, 563, 564, 565, 566, 567, 568, 569, 570, 571, 572, 573, 574, 575, 576, 577, 578, 580, 581, 582, 583, 584, 585, 586, 587, 588, 589, 6, 7, 8, 9]

Cluster: 2 ; n_gestures: 2 ;
Gestures: [257, 261]

Cluster: 3 ; n_gestures: 1 ;
Gestures: [18]

Cluster: 4 ; n_gestures: 1 ;
Gestures: [579]

g. DTW gesture-gesture similarity matrix Spectral clustering results:

Spectral Clustering Output:
min_avg_dist: 0.0032783063652781644 , n_random_starts: 100

Cluster: 1 ; n_gestures: 1 ;
Gestures: [579]

Cluster: 2 ; n_gestures: 1 ;
Gestures: [16]

Cluster: 3 ; n_gestures: 1 ;
Gestures: [21]

Cluster: 4 ; n_gestures: 90 ;
Gestures: [1, 10, 11, 12, 13, 14, 15, 17, 18, 19, 2, 20, 22, 23, 24, 249, 25, 250, 251, 252, 253, 254, 255, 256, 257, 258, 259, 26, 260, 261, 262, 263, 264, 265, 266, 267, 268, 269, 27, 270, 271, 272, 273, 274, 275, 276, 277, 278, 279, 28, 29, 3, 30, 31, 4, 5, 559, 560, 561, 562, 563, 564, 565, 566, 567, 568, 569, 570, 571, 572, 573, 574, 575, 576, 577, 578, 580, 581, 582, 583, 584, 585, 586, 587, 588, 589, 6, 7, 8, 9]

Spectral Clustering Output with Sklearn's KMeans:
Cluster: 1 ; n_gestures: 90 ;
Gestures: [1, 10, 11, 12, 13, 14, 15, 17, 18, 19, 2, 20, 22, 23, 24, 249, 25, 250, 251, 252, 253, 254, 255, 256, 257, 258, 259, 26, 260, 261, 262, 263, 264, 265, 266, 267, 268, 269, 27, 270, 271, 272, 273, 274, 275, 276, 277, 278, 279, 28, 29, 3, 30, 31, 4, 5, 559, 560, 561, 562, 563, 564, 565, 566, 567, 568, 569, 570, 571, 572, 573, 574, 575, 576, 577, 578, 580, 581, 582, 583, 584, 585, 586, 587, 588, 589, 6, 7, 8, 9]

Cluster: 2 ; n_gestures: 1 ;
Gestures: [16]

Cluster: 3 ; n_gestures: 1 ;
Gestures: [579]

Cluster: 4 ; n_gestures: 1 ;
Gestures: [21]

The results demonstrate that spectral clustering for most of the user options led to the same cluster assignments for both the implementations. Another observation that can be made about the minimum average distance of all points from their cluster centroids is that, in KMeans the minimum average distance is much larger as compared to that in Spectral clustering for the same gesture-gesture similarity matrices. Due to this, we might say that spectral clustering converges to the global minima, however, it still did not help in distinguishing among the 3 class data.

Interface Specification:

This system is a command-line application written in python. It supports 4 major tasks along with 2 subtasks:

1. **Task 0a:**
   Gesture feature/word extraction into a '.wrd' file for each gesture file. The script for this task is written in the task0a.py file in the code directory.
2. **Task 0b:**
   Generates TF and TF-IDF vectors for each gesture file in the dataset. The script for this task is written in the task0b.py file in the code directory. The output for this task is created in the data directory in the format tf_vectors_$file.txt and tfidf_vectors_$file.txt for each file in the dataset.
3. **Task 1:**
   Identifies the top-k latent semantics/topics, using user options PCA, SVD, NMF and LDA for a user specified model and k value. The output is presented in a json file in the form of <word, score> pairs in decreasing order of their scores for each of the k latent topics. The script for this task is written in the task1.py file in the code directory and the output file is phase_2_task_1_output.json.
4. **Task 2:**
   Given a query gesture object from the dataset, retrieves its top 10 most similar objects using user specified TF or TF-IDF model and supports 7 user options for dot product, PCA, SVD, NMF, LDA, edit distance and DTW. The script for this task is written in the task2.py file in the code directory and the results are presented in the console itself.
5. **Task 3:**
   Generates a gesture-gesture similarity matrix for the above 7 user options and reports the top-p principle components or gesture membership underlying the gesture-gesture similarity matrix using SVD and NMF. The script for this task is written in task3.py in the code directory and the output of this task is presented in the phase_2_task_3_SVD_output.json and phase_2_task_3_NMF_output.json files.
6. **Task 4ab:**
   Partitions the ‘n’ gesture objects into p groups based on their degree of membership to the top-p latent semantics of the gestures obtained using the task3. The script for this task is written in task4ab.py and the output is presented in the console.
7. **Task 4cd:**

Clusters the ‘n’ gesture objects into p groups based on user input using the gesture-gesture similarity matrix generated in task 3; where task4c uses KMeans algorithm for clustering and task4d uses Spectral clustering technique. The script for this task is written in the task4cd.py file and the output or cluster assignments are displayed in the console.

System Requirements:

- Operating System: Windows 10
- Python Version: 3.8.0
- Libraries Used: os, json, glob, sys, pandas, numpy, scipy, copy, math, ast, pickle, sklearn
- Setup Instructions:
    - Run the following commands to create a virtual environment name *demo* and install all of the dependencies
      $ python -m venv demo
      $ source demo/bin/activate
      $ pip install -r requirements.txt

Installation Instructions:

1. To run the program for Task 0A, execute the following command:
   $ python task0a.py <Directory> <Resolution> <Window Length> <Shift Length>
   Directory - The path to the dataset
   Resolution - An integer value
   Window Length - An integer value
   Shift Length - An integer value
   For example: $ python task0a.py ./test 3 3 3
2. To run the program for Task 0B, execute the following command:
   $ python task0b.py <Directory>
   Directory - The path to the dataset
   For example: $ python task0b.py ./test
3. To run the program for Task 1, execute the following command:
   $ python task1.py <Directory> <Vector Model> <User Option> <k>
   Directory - The path to the dataset
   Vector Model - ‘A string value (‘tf’ or ‘tfidf’)
   User Option - An integer value (Between 1 and 4)
   k - An integer value
   For example: $ python task1.py ./test tfidf 3 4
4. To run the program for Task 2, execute the following command:
   $ python task2.py <Directory> <Gesture File> <Vector Model> <User Option> <k>
   Directory - The path to the dataset
   Gesture File - An integer value

Vector Model - ‘A string value (‘tf’ or ‘tfidf’)
User Option - An integer value (Between 1 and 7)
k - An integer value
For example: $ python task1.py ./test 10 tfidf 3 4

5. To run the program for Task 3, execute the following command:
   $ python task3.py <Directory> <Vector Model> <User Option> <p> <k>
   Directory - The path to the dataset
   Vector Model - ‘A string value (‘tf’ or ‘tfidf’)
   User Option - An integer value (Between 1 and 7)
   p - An integer value
   K - An integer value
   For example: $ python task1.py ./test tfidf 3 4 4
6. To run the program for Tasks 4a and 4b, execute the following command:
   $ python task4ab.py
   For example: $ python task4ab.py
7. To run the program for Tasks 4c and 4d, execute the following command:
   $ python task4cd.py <User Option> <p> <Task>
   User Option - An integer value (Between 1 and 7)
   p - An integer value
   Task - A string value (‘C’ or ‘D’)
   For example: $ python task4cd.py 3 4 C

Related Work:

- A Tutorial on Spectral Clustering: Technical Report No. TR-149
  Max Planck Institute for Biological Cybernetics
  Author: Ulrike von Luxburg
  In this technical report[4], Ulrike gives a very straightforward description of what spectral clustering is and how we go about implementing this method. This report introduces us to the basic mathematical concepts and objects used in spectral clustering like similarity graphs, adjacency matrix, Laplacian matrix, etc. The report explains how these objects are created and offer additional information about the different parameters involved in the algorithm. The author mainly focuses on the creation of the Laplacian matrix and its properties, whose eigenvectors and corresponding eigenvalues contribute to the interpretation of the connectivity of the nodes. For the spectral clustering section, the report concludes by giving a normalized and unnormalized version of the algorithm and an example to show the difference.

- Latent Dirichlet Allocation
  Authors: David M Bei, Andre Y. Ng, Michael I. Jordan
  The authors describe a generative probabilistic model for collections of discrete data such as text corpora(or documents). The report introduces us to the concepts of PCA, which works on discrete data and the drawbacks of pLSI, which is usually described as a generative model, its documents have no generative probabilistic semantics. We are

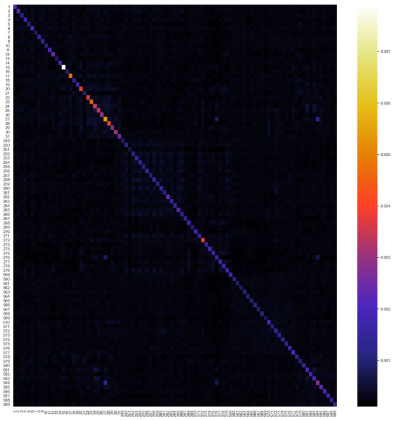
introduced to LDA, which is a new model for collections of discrete data that provides full generative probabilistic semantics for documents.

- Using Dynamic Time Warping to find patterns in time series
  Authors: Donald J. Berndt, James Clifford
  This paper describes preliminary experiments with Dynamic Programming approach to the problem of knowledge discovery in databases. The paper provides a brief explanation of Dynamic Time Warping. The paper describes a simple experiment to demonstrate the working of Naive DTW for looking for mountain patterns. The authors emphasize the importance and prevalence of inherent temporality in many types of data like stock etc.

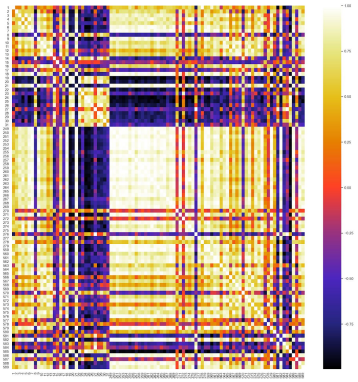
Observations:

- ❏ During our analysis of tf and tf-idf values and see if there is any correlation between the data in different dimensions, we observed that all the values of tf and tf-idf are coming out to be equal. Upon further analysis, we saw that the data in the files is equal across all the dimensions. That is the same data is replicated across all 4 dimensions

- ❏ **TF-IDF- dot product:** We observe below the gesture-gesture similarity graph obtained using dot product and tf vector model. We observe that the highest match is observed along the diagonals which is consistent because diagonals represent similarity between the same documents which are naturally the highest.

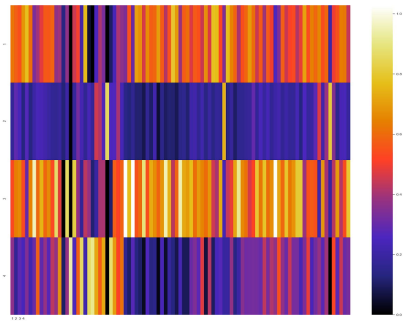


- **TF IDF- PCA:** For the similarity graph obtained using PCA as well we are observing a

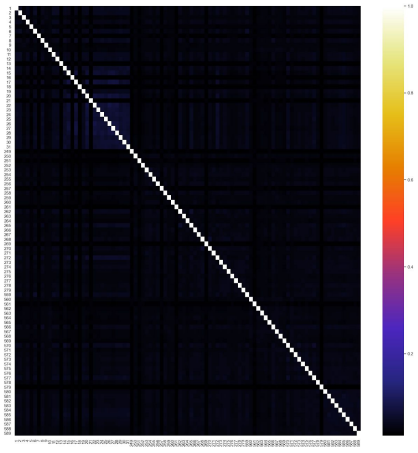


similarity between files. From the image above We can notice that the files in the range 1-15 are similar to each other but not similar to the files 15-30. The same is true for the files in range 15-30. This may be because files 15-30 be a reverse of file 1-15

- **TF-IDF SVD's SVD top p semantics:** Performed SVD on SVD's gestural-gesture similarity matrix. As we can see, the top 2 semantics and their contribution to the similarity matrix.



- **Edit Distance(No need of vector model):**
  Edit distance algorithm shows trends on expected lines. The highest similarity is observed along the diagonal. The elements around the diagonal show higher similarity than the ones off them. The first one third of the graph represents the first set of gestures, and correspondingly we can see a cluster of purple color around the first one third part of the graph close to the top-left corner. Similar trend is observed for the other two gestures.



Conclusions:

After observing the above results for every task, we saw the effectiveness and usefulness of dimensionality reduction techniques on time-series data. A few takeaways could be that using methods like PCA, SVD, NMF, etc. could result in a possible loss of information. We could see that using distance metrics like Edit Distance and DTW could marginally improve results because they consider the underlying temporal data, which is ignored in metrics like dot product or cosine similarity. We learned that the transformed matrix from LDA is a Probability Distribution Function, hence, it is better to use metrics like K-L Divergence to compare two gestures. Although the results from the clustering tasks were a bit confusing, our algorithm was able to match the output to the inbuilt library methods. Finally, we would like to conclude our report by thanking Prof Candan for this project.

Bibliography:

1. PCA: https://builtin.com/data-science/step-step-explanation-principal-component-analysis
2. KMeans: https://towardsdatascience.com/clustering-using-k-means-algorithm-81da00f156f6
3. Spectral clustering: https://towardsdatascience.com/spectral-clustering-82d3cff3d3b7
4. Von Luxburg, Ulrike. "A tutorial on spectral clustering." Statistics and computing 17.4 (2007): 395-416.
5. NMF: https://mlexplained.com/2017/12/28/a-practical-introduction-to-nmf-nonnegative-matrix-factorization/
6. DTW: https://www.aaai.org/Papers/Workshops/1994/WS-94-03/WS94-03-031.pdf
7. LDA:
    a. https://en.wikipedia.org/wiki/Latent_Dirichlet_allocation
    b. https://towardsdatascience.com/latent-dirichlet-allocation-lda-9d1cd064ffa2
    c. Latent Dirichlet Allocation - David M. Blei, Andrew Y. Ng, Michael I. Jordan[https://www.jmlr.org/papers/volume3/blei03a/blei03a.pdf}

Appendix:

Amey Athale:

- Code contributions: Task 0a, Edit distance function, Spectral clustering
- Report contributions: Abstract, Goal description, Proposed solution - Task 0a and Edit Distance, Related Work, Conclusion

Vineeth Chitteti:

- Code contributions: Task 1, Task 3, Task 2
- Report contributions: Task 1

Shreesh Nayak:

- Code contributions: Task 0a, Task 3a and Task 3b, Integration of code, Heatmaps
- Report contributions: Task3, System requirements, Heatmaps, Installation instructions

Lokesh Sharma:

- Code Contribution: Task 0b, DTW distance algorithm, Task 4a and Task 4b, heatmap
- Report Contribution: Terminology, Task 0b,Task 2(DTW), Task3(DTW), Task 4a, Task 4b, heatmaps, Observations.

Ankit Kumar Rai:

- Code Contributions: Tasks 0b, 1, 2, 4a, 4b
- Report Contribution: LDA, Spectral clustering, Task 2, Task 3, terminologies, related work, observations,

Richa Nagda:

- Code Contribution: Task 0a, KMeans implementation
- Report Contribution: Assumptions, PCA, Task 1, Task 2, Task 4c, Interface Specifications