CSE 515 Group Project Phase 1

Michael Cai
Justin Zoratti
Matthew Budiman
Ahmed Usman
Dalton Turner
Shivam Sadachar

September 20th, 2020

Abstract

In this project we pulled two datasets from Chalearn containing multivariate time series data of gestures and examined how we could represent them in vector models. The data was normalized and quantized to create word representations for various points in time. The TF and IDF values were calculated for these words and used to make vector representations of the gestures. A utility was created to view these vectors in the form of a heatmap. We then demonstrated how these vector representations could be utilized to compare gestures enabling us to search for similar gestures given a query.

Keywords

Time-series, multivariate, term-frequency, inverse-document-frequency, vector-model, heatmap, gesture, sensor, querying, database

Introduction

The goal of this project was to represent multivariate time series data in a model such that we could visualize it and query on it utilizing other time series objects as query objects. For this particular phase we chose to represent the data as vector models. We downloaded the Montalbano V1 (ICMI ’13) and V2 (ECCV ’14) datasets from Chalearn. The original input data is in the form of several csv files which contain the positional information of motion tracking sensors. This time series of movements is referred to as a gesture throughout the rest of this report. The gestures are recorded using a variety of sensor inputs on different parts of the person’s body over time, the sensor’s positions are represented by floating decimal values.[5] Within each gesture we categorized the values into different bins based on the similarity of values. These bins will be referred to as bands containing a lower and upper bound for a value to fall within the band. While the data is being processed the floating point values are replaced with the corresponding band they have been categorized into. The data is then used to create words, a representation of the series of data beginning at a period in time quantized by the band they are in so values across gestures can be compared.[4]

The window size refers to the amount of these quantized values that are put together to create a word. The shift value refers to how much time is skipped from the starting position of the last word to create the next word. These vector models are vectors in which each position corresponds to a unique word seen throughout the processed data and the values in each index are either the TF, TF-IDF, or TF-IDF2 values.[4] TF or otherwise known as term frequency is a value that indicates the weight of a word in one gesture-sensor pair. IDF refers to the inverse document frequency which is indicative of the inverse of how many gesture-sensor pairs a word occurs. TF is multiplied by IDF to create TF-IDF.[2] In TF-IDF2, IDF is modified so only the words in the given gesture are considered.[4] This measured the discrimination power among sensors in a single gesture.

Finding similar objects in time series data can be difficult in its raw form because it’s difficult to find measures by which the objects can be compared. It’s possible that the time before and after the gesture varies by file and that the gesture is performed more dramatically in one example compared to another. We created the program with the assumption that the gesture could occur at any time during the file. We also assumed that the data would be completely present and we would not have to account for any missing data in a gesture file such as a missing sensor or time value.

Implementation Description

For each of the tasks of the project a python file was created to implement the solution. For file reading purposes, an additional file *iohelpers.py* helps with reading some of the files the program creates throughout the project. All output from each of the python files is performed in the same folder as the source data. For the purposes of a document we defined it as being a gesture-sensor pair in our proposed solution. That is the unique tuple of the gesture and sensor would be considered one object when calculating our TF and IDF values.

Task 1

For this task of the project we created a vector representation of multivariate time series data by extracting the data and classifying it into groups. We normalized the gesture documents ensuring the highest value for a sensor in a gesture document was 1 and the lowest was -1 with all other values proportionately adjusted. We chose to normalize on a gesture document instead of the full dataset due to the fact a gesture may be performed not in the center frame of the motion tracking system. This would cause a gesture that was performed in a slightly different position to be categorized as different words if we normalized by directory. Therefore we chose to normalize by gesture. The length of bands bands were calculated using the following equation where r is the resolution inputted by the user:

$$length_i = 2 * \frac{\int_{(i-r-1)/r}^{(i-r)/r} Gaussian_{(\mu = 0,\, \sigma = 0.25)}\, x\, d(x)}{\int_{-1}^{1} Gaussian_{(\mu = 0,\, \sigma = 0.25)}\, x\, d(x)}$$

We generated 2*r band lengths and created cutoffs centered around 0. We then utilized these cutoffs to quantize the original data.[4] This allowed us to convert the data from a time series of continuous location data from sensors to a time series of discrete bands calculated from the normalized values of the raw data. These bands caused the loss of some information but gave us a way to efficiently store the time series data as a set of words with some metadata information regarding the gesture, sensor, and time they belonged to. The new quantized time series data is stored in a wrd file with its associated metadata for each word.

Task 2

Utilizing the output of task 1, each gesture-sensor pair is converted into three different gesture vectors measuring the TF, TF-IDF, and TF-IDF2. The following equations were utilized for the various calculations of TF and IDF[2]:

$$TF = n / K$$
$$IDF = log(N / m)$$

The variables refer to:
n - The number of times a particular word shows up in a gesture-sensor pair across all gestures.
K - The total number of words in a gesture-sensor pair.
N - The number of gesture-sensor pairs total.
m - The number of gesture-sensor pairs that contain that particular word.[2]

N and m are calculated utilizing different scopes for IDF and IDF2. In IDF we consider all gesture-sensor pairs across all gestures, while in IDF2 only the gesture-sensor pairs for the same gesture are considered.[4]

These vectors are created such that each unique word seen across all gestures is assigned an index within a vector. For each gesture-sensor pair 3 of these vectors are assigned where they contain the TF, TF-IDF, and TF-IDF2 values for their respective words. These vectors are then saved in the file *vectors.txt.*

Task 3

Utilizing the wrd files and *vectors.txt* information, we generated heat maps displaying the user's choice of TF, TF-IDF or TF-IDF2 values for a particular gesture. We mapped each sensor for a gesture to a particular row and each point in time in which we recorded each word on the columns. At the intersection of these is a greyscale box which indicates the values chosen by the user. This scale is indicated on the right of the heatmap. These heatmaps are only displayed during the duration of the program and in order to save an image of them the matplotlib save function must be utilized on the display window.

Task 4

Also utilizing the wrd files and *vectors.txt* information, we implemented a way for the user to input the name of one of the previously processed gesture files and have it compared to the rest of the files in order to find the 10 most similar gestures. The user also inputs their choice, the measure in which all of the gesture files will be compared between the TF, TF-IDF, or TF-IDF2 values of each gesture. The vectors from *vectors.txt* are then compared to the user's gesture file input. For this comparison we utilized the Euclidean distance between two vectors to be able to accurately measure the similarity of the vector model objects.[3] The Euclidean distance gave a simple means of measuring the distance between our preprocessed data. Once the euclidean distance for each sensor was found we were left with a vector of distances between the gestures. We chose to calculate the magnitude of this vector to determine the final distance value as a floating point decimal.

$$\sqrt{\sum_{i=1}^{n} |q_i - o_i|^2}$$

The formula for Euclidean distance between two vectors

Interface Specifications

This program is intended to extract data from several *.csv files containing the motion data from sensors recording a gesture. Each of these files has each sensor tied to one row of the data and each column is a particular sensor's recording of the person's position at a particular time. In order for the program to run currently the input must contain all of the sensor's recordings for a particular point in time, there must be no missing data points. There can be as many time periods (columns) recorded as the user wants.

Task 1

The program gestureWords.py takes the csv files containing the gestures as input and transforms them into wrd files. There will be one wrd file generated for each csv, with the name before the file extension being the same as its corresponding csv. The contents of the wrd file are several lines containing metadata for each word encountered in the csv. The metadata contains the name of the file it was taken from, the sensor id (0-indexed) that it was taken from, and the time in which the start of the word occurs. These wrd files are saved in the same directory as the csv files. The wrd files for test1 - test6 are present in the outputs folder.

$$< fileName,\ sensorid,\ time >,\ \ < word >$$

The format of each line in a wrd file as defined in project specifications[4]

[['test1', 9, 38], [1, 3, 3]]

An example representation of the word [1,3,3] from gesture test1, sensor 9, starting at time 38

Task 2

The second program, createVectors.py, builds off the wrd files created by task 1. The wrd files are processed into TF, TF-IDF, and TF-IDF2 vectors. These vectors are stored in the file *vectors.txt* in the same directory as the wrd files it was created from. The file is a newline separated file for each gesture-sensor pair with the exception of the first line. The first line contains a list which indicates the positions of each encountered word in the vectors. Besides this first line every other line is a hash separated value. The first value is the gesture-sensor pair it is for with the second through fourth being for its TF, TF-IDF, and TF-IDF2 vectors respectively. This allows a program to break apart the data easily into nested dictionaries and extract information.

For example if the TF value of 4,4,4 wanted to be determined for gesture-sensor (3, 2) then the first line would be consulted to find the index of 4,4,4,. Afterwards the line containing (3,2) as its first hash separated value would be found, the second hash separated value would be (3,2)'s TF vector. Then the vector could be indexed at our originally found index of 4,4,4. This format allows us to save space by not storing the indexing for each and every sensor's unique set of words. It also makes comparing the vectors easier task 4 when it comes time to compare the distances between them.

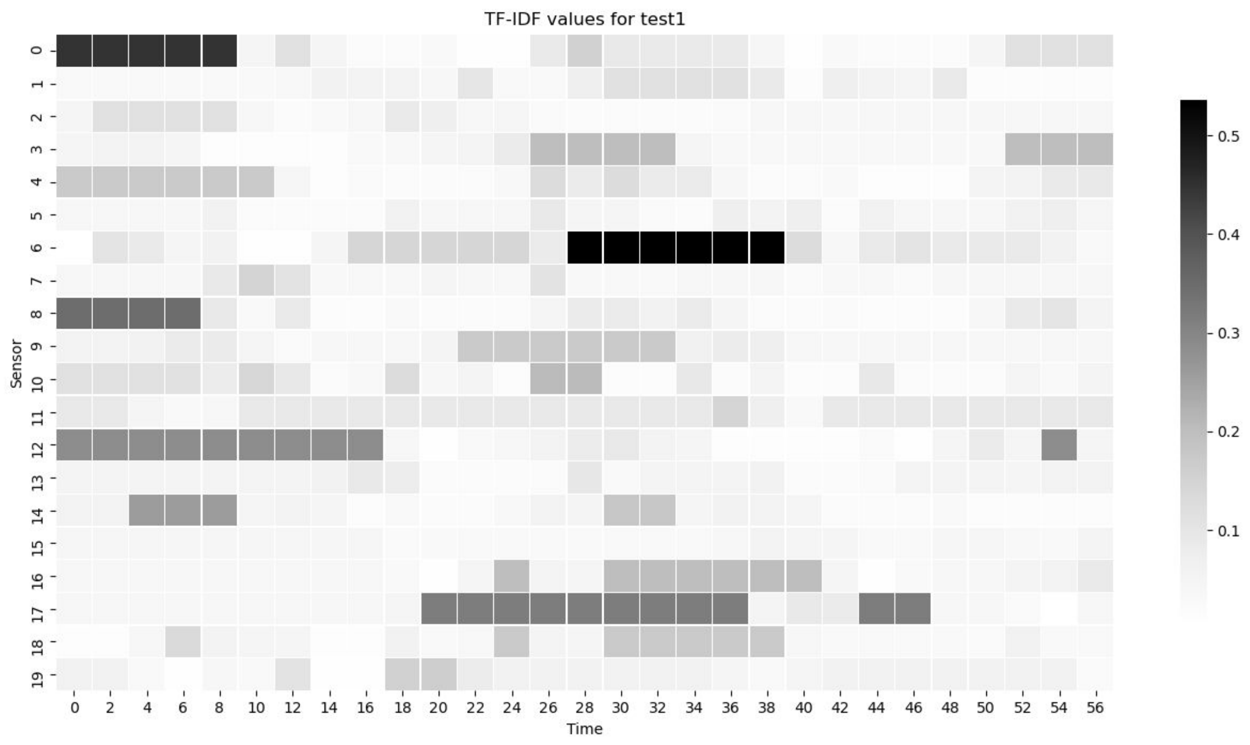
[(5, 5, 5), (5, 4, 4) ...]
('1', 0)#[0.263, 0.053 …]#[0.220, 0.064…]#[0.210, 0.073...]

Example of data vectors.txt, data has been omitted with ellipses and rounded for succinctness.

In the above example the first line shows the indexes of (5,5,5) and (5,4,4) as 0 and 1 respectively. We can also see the corresponding values in the TF, TF-IDF, and TF-IDF2 vectors for those words for gesture 1, sensor 0. The *vectors.txt* for all 66 items in the database is present in the outputs folder.

Task 3

The third program heatmap.py reads both the wrd files and *vectors.txt* file to create a heatmap. Both are automatically read from the inputted directory. The final output is not automatically saved to disk but can be saved using matplotlib's built in figure saving function.

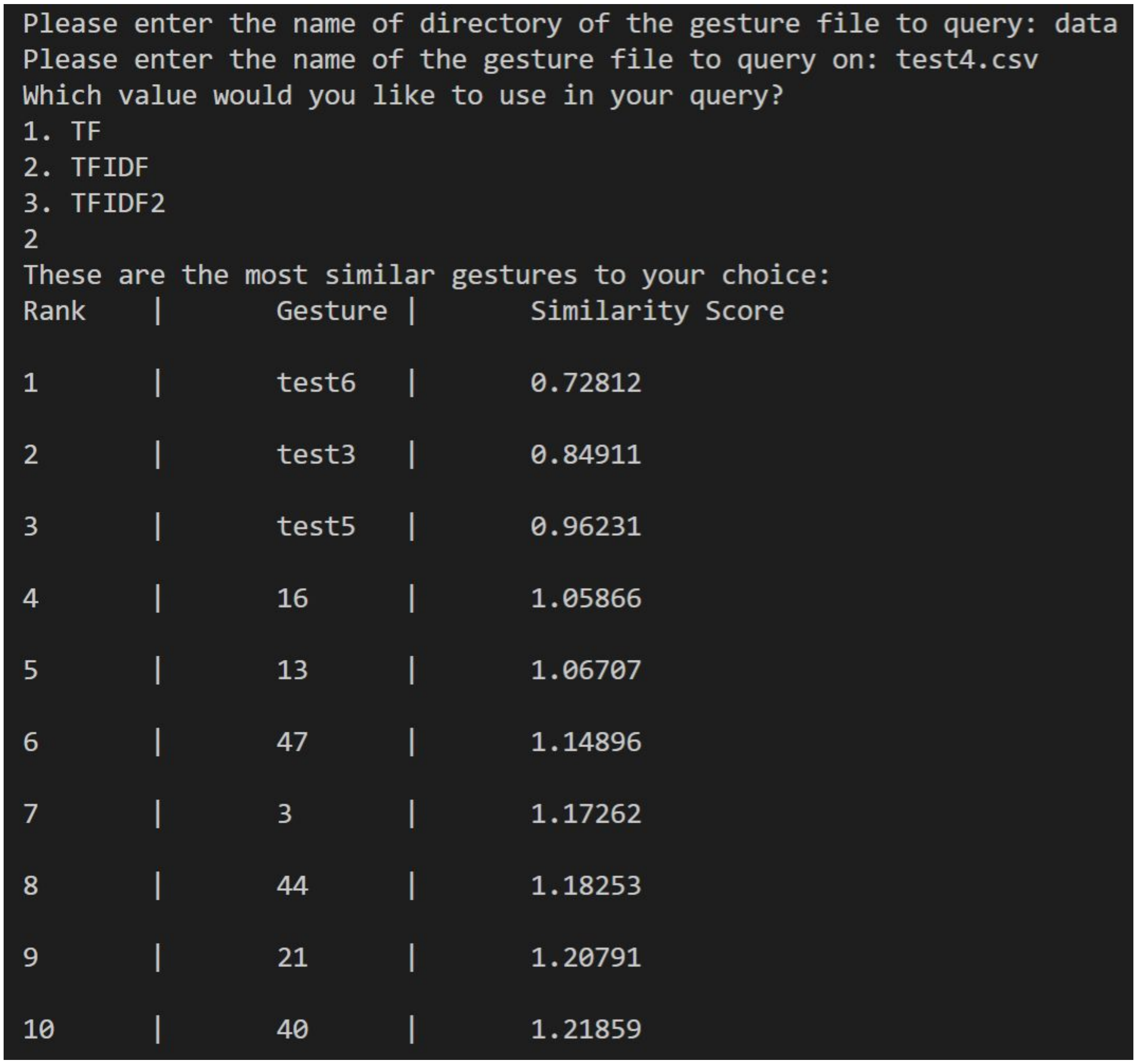


An example heatmap for the TF-IDF values of gesture test1 with r = 3, w = 3, s = 2

In the above example we can see the TF-IDF outputs for test1.csv. This heatmap shows the defining words of this gesture occurred on sensor 0 at time 0-8 and on sensor 6 at time 28-38. These are sequences of words that are most distinct within test1.csv and querying on gestures containing those words would likely return test1.csv as one of the results. It may not be ranked highly however as our maximum for TF-IDF values is ~1.78 meaning that these words are not exclusive to test1.csv. The heatmaps for test1 - test6 are included as a part of the outputs folder.

Task 4

The fourth program query.py is run using the *vectors.txt* file to determine the most similar gestures to an inputted gesture. The vectors file is automatically read from the inputted directory from where the gesture is. The output from this task is printed directly in the command line console. The results are printed in three columns displaying the ranking of the result, the gesture name without the file extension, and the distance between the two files utilizing our distance function. The results from querying on test1 - test6 are included in the outputs folder as a part of *queryResults.txt.* The name of the file queried and the associated value to query on is put before each of the corresponding results.



```
Please enter the name of directory of the gesture file to query: data
Please enter the name of the gesture file to query on: test4.csv
Which value would you like to use in your query?
1. TF
2. TFIDF
3. TFIDF2
2
These are the most similar gestures to your choice:
Rank    |       Gesture |       Similarity Score

1       |       test6   |       0.72812

2       |       test3   |       0.84911

3       |       test5   |       0.96231

4       |       16      |       1.05866

5       |       13      |       1.06707

6       |       47      |       1.14896

7       |       3       |       1.17262

8       |       44      |       1.18253

9       |       21      |       1.20791

10      |       40      |       1.21859
```

Example output from the query on test4 comparing TF-IDF values

In the example above we can see that the most similar gesture to test4 is test6. The magnitude of all of their sensor Euclidean distances is the least compared to all other files.

System Requirements / Instructions

Pre-requisites:

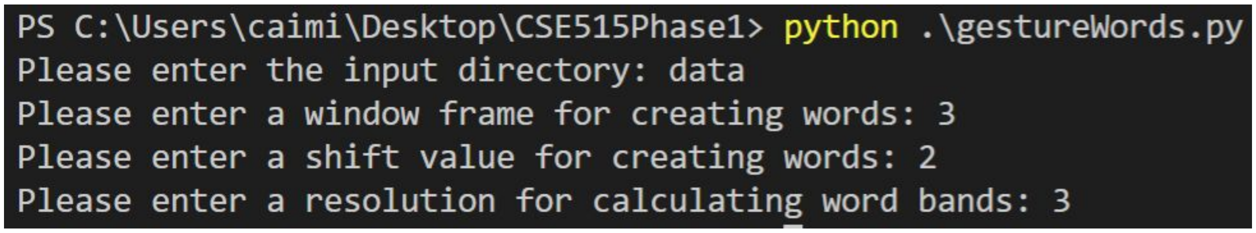
- Python 3.8.5
- numpy 1.19.2
- pandas 1.1.2
- seaborn 0.11.0
- matplotlib 3.3.1

The program is a set of python files which execute each task. The python files and their corresponding tasks are:
Task 1 - gestureWords.py
Task 2 - createVectors.py
Task 3 - heatmap.py
Task 4 - query.py

These programs are intended to run in the order of their tasks.

gestureWords.py

The program can be run with the command `python gestureWords.py` from the command line. It will prompt you asking for the name of the directory where all the *.csv gesture files are located. Then it will ask for three integer values, the word window frame size, the shift amount for word windows, and the resolution to store data respectively. Once all inputs are specified the program will process all files in the given directory into *.wrd files and save them in the same directory.

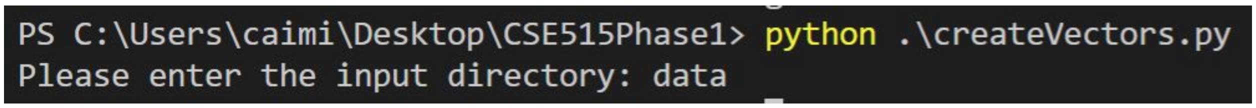


```
PS C:\Users\caimi\Desktop\CSE515Phase1> python .\gestureWords.py
Please enter the input directory: data
Please enter a window frame for creating words: 3
Please enter a shift value for creating words: 2
Please enter a resolution for calculating word bands: 3
```

Example input for gestureWords.py

createVectors.py

The program can be run with the command `python createVectors.py` from the command line. It will take a singular input, the directory containing the *.wrd files generated from *gestureWords.py.* These word files will be converted into vectors that contain the TF, TF-IDF, and TF-IDF2 values. These vectors are stored in the same directory as the read word files in a file named *vectors.txt.*

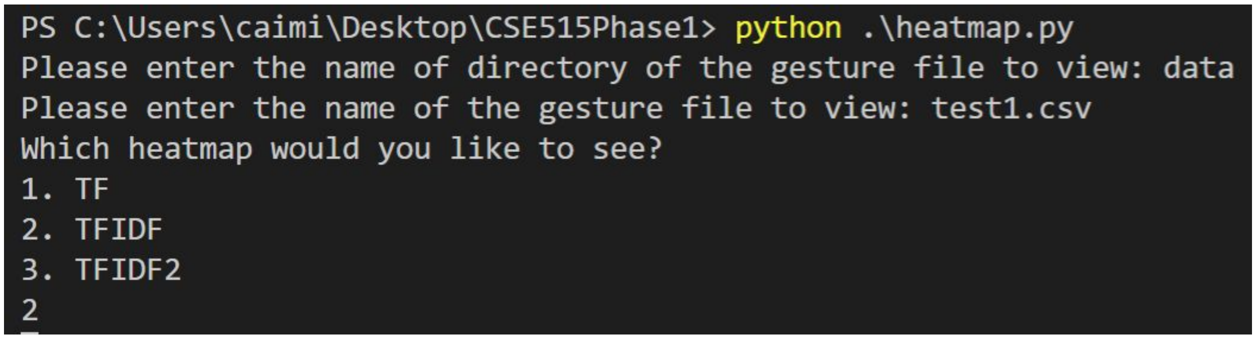


```
PS C:\Users\caimi\Desktop\CSE515Phase1> python .\createVectors.py
Please enter the input directory: data
```

Example input for createVectors.py

heatmap.py

The program can be run with the command `python heatmap.py` from the command line. It will take three inputs: the directory containing the gesture file, the name of the original gesture file csv (this will include the .csv file extension), and a numerical choice as to what values the heatmap should display (1 - TF, 2 - TF-IDF, 3 - TF-IDF2). The program will display a figure using matplotlib which contains the heatmap. The sensors of the gesture are listed on the y-axis plotted against time on the x-axis. A scale of the values is shown on the right with each value displayed as a square in the corresponding location. To end the program close out of the displayed figure.

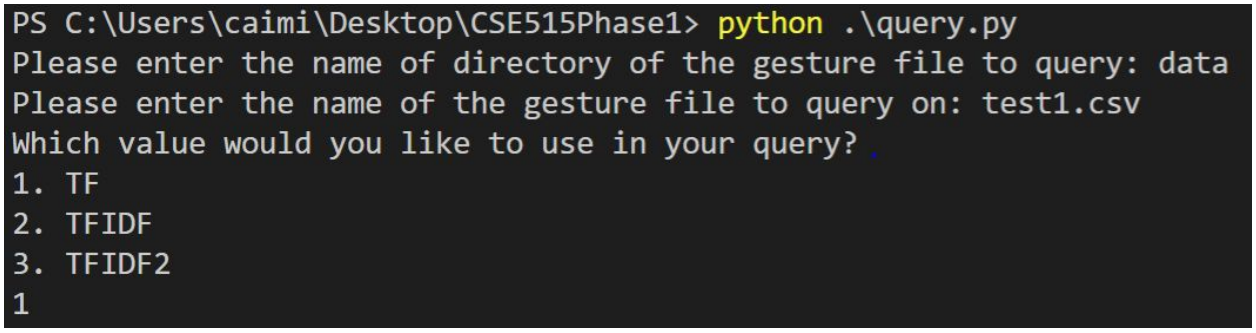


```
PS C:\Users\caimi\Desktop\CSE515Phase1> python .\heatmap.py
Please enter the name of directory of the gesture file to view: data
Please enter the name of the gesture file to view: test1.csv
Which heatmap would you like to see?
1. TF
2. TFIDF
3. TFIDF2
2
```

Example input for heatmap.py

query.py

The program can be run with the command `python query.py` from the command line. It will take three inputs: the directory containing the gesture file, the name of the original gesture file csv (this will include the .csv file extension), and a numerical choice as to what values the search should consider when querying (1 - TF, 2 - TF-IDF, 3 - TF-IDF2). The program will find the 10 most similar gestures to the inputted file and print to the console a list in which their similarity rank, file name (without .csv attached), and similarity score are listed. The similarity score functions as a measure of distance so more similar files have lower similarity scores.



```
PS C:\Users\caimi\Desktop\CSE515Phase1> python .\query.py
Please enter the name of directory of the gesture file to query: data
Please enter the name of the gesture file to query on: test1.csv
Which value would you like to use in your query?
1. TF
2. TFIDF
3. TFIDF2
1
```

Example input for query.py

Related Work

This program is a rudimentary form of representing time series data in a database to be queried and visualized. Many other solutions for representing time series have been created such as the data historian model. data Historians can store data in both a lossy and lossless format, although they keep the data in the time-series format instead of converting to vector models. In order to help make information comparable they use a schema which involves the use of tags.[6] We utilized a version of these in which we recorded the sensor that the data came from, but data historians tend to record more tags in the form of user-defined data or events.

The Chalearn website and other papers also seem to support that neural networks serve as a good way of storing gesture information especially when the data is multimodal.[1] They would be able to encode more data than our vector representations and have a very high accuracy when determining similar gestures given a query.[5] However training time for these neural networks might take much longer than the recalculation of norms, TF, and IDF values in our vector representation making it difficult to add data.

Conclusions

This phase of the project showed how complex data can be organized into a form in which it is compressed to lose some information but becomes much more efficient to load and query. Time series data is often difficult to manage especially if there are multiple variables all operating on different quantitative scales. By saving a form of data which essentially highlights the most important and defining words for a singular document (gesture-sensor pair in our case) we could compare it to other objects with ease. The heat maps generated in task 3 allowed us to visualize just how particular words can be extracted as defining characteristics exclusive to certain gestures and become the features that allow us to differentiate sets of documents only containing those words. We put those values in action by implementing a way to query on them and determine similar gesture files. We lost some temporal information during the creation of this model which makes our implementation imperfect in the end but serves its purpose as a basic way of representing time series data.

Bibliography

1. B. Zhao, H. Lu, S. Chen, J. Liu and D. Wu, "Convolutional neural networks for time series classification," in Journal of Systems Engineering and Electronics, vol. 28, no. 1, pp. 162-169, Feb. 2017, doi: 10.21629/JSEE.2017.01.18.
2. Candan, K. S., & Sapino, M. L. (2010). Feature Quality and independence. In *Data management for multimedia retrieval* (pp. 146-148). Cambridge: Cambridge University Press.

3. Candan, K. S., & Sapino, M. L. (2010). Comparison of Objects in the Vector Space. In *Data management for multimedia retrieval* (pp. 102-104). Cambridge: Cambridge University Press.
4. Candan, K. S. (2020). CSE 515 Multimedia and Web Databases Phase #1 Project Description (pp. 1-2). Published on Piazza
5. ChaLearn. (2013). Multimodal Gesture Recognition: Montalbano V1 (ICMI '13). Retrieved September 20, 2020, from http://chalearnlap.cvc.uab.es/dataset/12/description/
6. Chardin, B., Lacombe, J., & Petit, J. (2013). Data Historians in the Data Management Landscape. *Selected Topics in Performance Evaluation and Benchmarking Lecture Notes in Computer Science,* 124-139. doi:10.1007/978-3-642-36727-4_9

Appendix

During this phase of the project each individual was responsible for implementing the program on their own. All six members did coordinate to help validate data outputs and discuss a variety of techniques for calculating intermediary values such as tf-idf values and query distances.
Michael Cai - Individual Project Code and Report
Justin Zoratti - Individual Project Code and Report
Matthew Budiman - Individual Project Code and Report
Ahmed Usman - Individual Project Code and Report
Dalton Turner - Individual Project Code and Report
Shivam Sadachar - Individual Project Code and Report