Data Mining and Decision Making

LINFO2275

Gestures Recognition

Group 4
Grognard Simon, 37811700, INFO
Guerrero Cano Juan Valentin, 36092200, EPL- IN
Marques Mathias, 13181700, INFO

February 21, 2024

Introduction

This report will compare several method concerning gesture recognition. In order to make these comparison, some datasets are needed, they are public and available on the internet [1]. The first dataset corresponds to ten people who have made ten trials of drawing the number from zero to nine.The second dataset concerns the drawing of 3D figures, with the same logic; ten people drawing ten trials of ten different figures. .

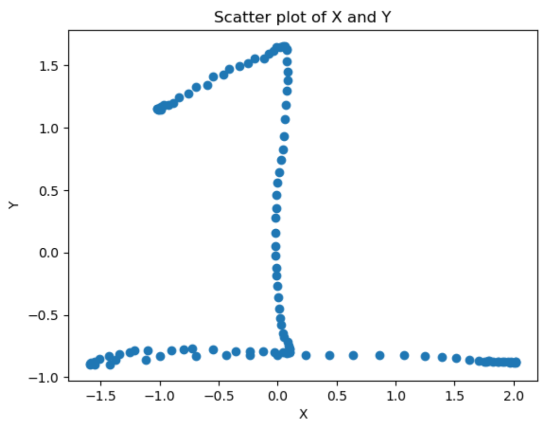
First of all, it is interesting to go through the different datasets, to have a better future approach of it. As the datasets is divided in 4D (the 3D poistions and the time), some preprocessing can be done in order to plot the numbers and the figures in two and three dimensions. Some python package allows to standardaize these data.

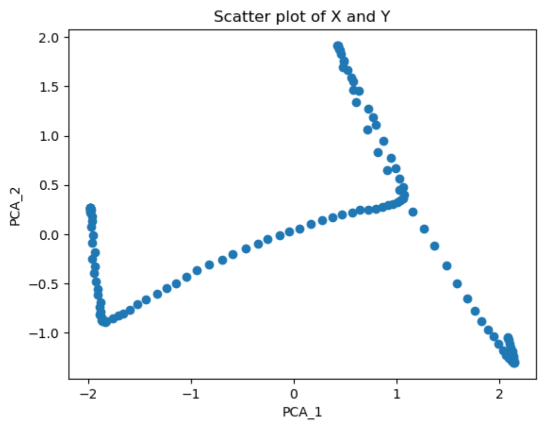
After that, it is time to make some recognition on these data. It exists some python's package but it is also interesting to implement these functions ourselves. In this report, we are going to talk about the implementation of edit-distance and about the $1 recognizer function.

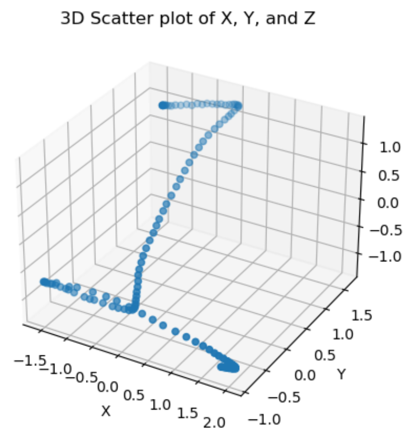
Finally, we will compare the difference between these methods using two tests, the user-independent and the user-dependent. In the first one, the different test sets are divided by user and the second one the test sets are data coming from all the users, one sample of each number of each user.

Exploration of the data

In order to gain a deeper understanding of the data we are working with, we decided to create plots. Below are examples of the plots we generated for the digit "one". We experimented with removing the z-coordinate to observe if the number remained recognizable. As depicted below, we found that the number could still be identified without the z-coordinate. To further refine our analysis, we conducted a PCA. To see if it was still possible to read the numbers and as you can see below it's recognizable. Finally,we also have plot the 3D points.







Baseline Method

Several approaches were suggested for this project. The chosen method involves implementing the edit distance methods, which require the implementation of a few preliminary functions before delving into this algorithm.

Pre-processing

The process begins by loading all the subjects and trials into a large 3D matrix. Next, the data is standardized using the StandardScaler function from sklearn. Initially, an attempt was made to perform a PCA, but this idea was dropped upon observing the initial results. Subsequently, it was found that the accuracy was significantly lower, while the computation time remained relatively unchanged. Additionally, it was decided to remove the time variable from the dataset since only the coordinates are sufficient for predicting the drawing.
Next, a 2D matrix is computed, containing all the coordinates (x, y, and z). A k-means clustering algorithm is applied to this matrix in order to obtain the clusters for each coordinates. Various values for the number of clusters were tested during this process as it will be shown in the result part. Then, a matrix is created for all the trials, where instead of the coordinates, the list of clusters corresponding to each coordinates is included.

Edit distance

The edit distance, is a metric used to measure the similarity between two strings. It quantifies the minimum number of operations required to transform one string into another. These operations include insertion, deletion, and substitution of individual characters.
To calculate the edit distance between two strings, you start with an empty matrix with dimensions (m+1) x (n+1), where m and n are the lengths of the two strings. Each cell in the matrix represents the edit distance between substrings of the two strings.
The algorithm fills in the matrix, starting by filling the first row and the first column. Then, at each cell, the algorithm considers three possible operations: insertion, deletion, and substitution. The edit distance at a particular cell depends on the values of the adjacent cells and the current characters being compared.
If the characters at the current positions in the strings are the same, no operation is needed, and the edit distance for the current cell is the same as the diagonal cell's edit distance. If the characters are different, the algorithm chooses the minimum edit distance from the adjacent cells and adds 1 to account for the substitution operation.
Once the entire matrix is filled, the edit distance is found in the bottom-right cell. This value represents the minimum number of operations needed to transform one string into the other.

After describing the implementation of the edit distance methods, let's explore how to utilize them in our specific case. The data will be split into a training set and a testing set. There are two types of testing methods employed. The first method involves removing a subject from the training set and using it as the test set. The second method involves removing a trial from each number for each subject. In both cases, the dataset is divided into 900 training samples and 100 testing samples. Subsequently, each test sample is compared with all the training samples, and the results of the edit distances are recorded in a 3D matrix.

Next, we proceed to select the k nearest neighbors from the 3D matrix (different value of k will be tested in the Validation section). The predicted label is determined by selecting the label with the highest frequency among the k nearest neighbors. In case of a tie between different labels, to break the tie, we consider the lowest sum of metrics among the options, which are grouped based on the most represented labels. If a tie still exists, one label is randomly chosen from the tied options.

In an attempt to improve our approach, an alternative method involved calculating the edit distance on substrings rather than entire strings. The idea behind is to compare string of the same length istead of different ones. However, this approach proved to be computationally expensive, and the resulting outcome did not yield better results. Consequently, the decision was made to not use this solution.

State-of-the-art Method

The second step of the project was to implement another model to be able to compare its results with the obtained results from the explained baseline method. We were suggested some different models. Our first idea was to implement a SVC using for training a precomputed kernel matrix using the pairwise distance. However since we got some problems while implementing this pairwise distance, we changed our idea to one totally different. After reading the doc from the dollar recognizer we thought that maybe the $1 recognizer could achieve good results from the aim of this part of the project. So finally, we decided to continue with the implementation of this recognizer. It is remarkable that for all the process to implement it we built different functions (each one properly commented) to achieve a clean code.

Pre-processing

Similarly to the preprocessing of the baseline method, we needed to do some preprocessing before implementing the recognizer. First, we read properly all the files we had, to being able to work with adequately python structures for our aim (in our case, mainly arrays and lists). For the different domains we also created the specific labels datasets, cause as we will explain later, we needed them to implement the $1 recognizer. Contrary to the baseline implementation, we reach, after some testing, the conclusion that for this recognizer the PCA was necessary. This PCA was implemented just taking two components.

$1 recognizer

Before explaining how we proceeded to implement it, we should explain first how this recognizer works. The $1 recognizer is a 2-D single-stroke recognizer, what means that the gesture is done in one time, and not with different strokes. Therefore, it fits perfectly our needs as our two domains are single-stroke gestures. This recognizer is, actually, a instance-based neares-neighbor classifier with the uclidean distance function. To be able to predict unseen data, this recognizer do some kind of preprocessing to the data, as resampling and normalization.

After reading the documentation of it, we search for a library where the $1 recognizer were implemented. We found the library "dollarpy" and it was based on three key elements. Point, which is actually a type of object that represent a 2D point with a third element representing the stroke (in our case this stroke were omittable, as we said before). Template, which is just a duple composed by a list of Point's objects, and the label that represent this list of points. Finally, the Recognizer, which just needed as hyperparameter a list of templates ("training set", which it will preprocess by its own as explained before), which it would use for predicting the labels of unseen list of Points. The way it works for predicting the labels is comparing the unseen list of points to the templates used for training the recognizer and giving to this comparison a score (using a similiarity scoring algorithm). Finally, it predicts the label just taking the comparison with the highest score.

After seeing some different examples of the use of the dollarpy library, we implemented several functions to use it properly. In order to do the split of the data for the two differents validation that the statements required us (user-independing and user-depending) we created carefully two functions. After, to perform the prediction, we split our whole dataset and labels into training and testing sets and then ”train” our recognizer with the templates created from the training set. Also it is remarkable that we needed to reformat data several times as templates needed specific types of parameters. Just after we predicted the labels for the testing dataset and compute the accuracy score. The results are shown in the following section.

Validation and comparison tests

In order to validate and compare the models, two kind of test have been made

- The user-independent: this test take 9 users to train the model and then it test it on all the trials of one user.
- The user-dependent: this test take one trial from each number from all users to test the model, the remaining data is for the training.

Edit distance method

The goal of testing is to find the better parameters to improve the accuracy. We have work on Two hyper parameters for the edit distance method. The first one is the number of clusters and the second one is the number of neighbors.

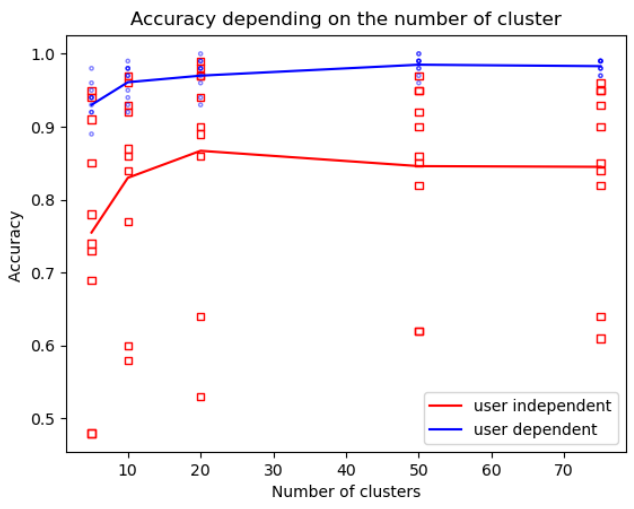


Figure 1: Accuracy of the two validations depending on the clusters

In this first test, we compare the two different test by modifying the number of clusters in the preprocessing. It is really easy to see that the user dependent has better accuracy for every cluster. The different point on the graph correspond to the different accuracy for each user according to the clusters. It is quite interesting that the points for the independent have a higher variance compare to the ones of the dependent.

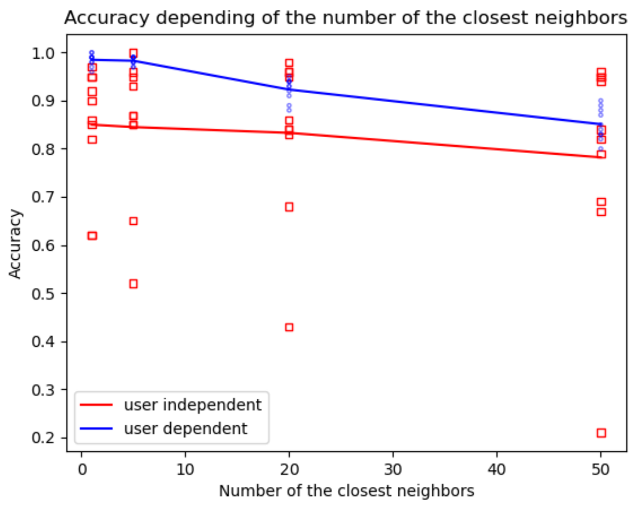


Figure 2: Accuracy of the two validations depending on the closest neighbors

In the second test, we still compare the different tests, however, in this case we change vary the k of the nearest neighbors using 50 clusters. The graph above show again that the dependent have better results then the independent and that the independent test have also a great variance compare to the dependent test.

A final test that have be done is about the domain 4, we train again the model using the edit-distance model with the two different settings. In this case we do not modify the parameter but use a cluster of 50 and we select the 1 nearest neighbors. The result for the user independent (resp. dependent) have an accuracy of 0.768 (resp. 0.96).

$1 recognizer

In other hand, for the $1 recognizer we do not have hyperparameters to tune, so we just implement the validation for this model got the following results:

For the first domain we got a mean accuracy of 0.967 with a variance of not more than 0.04 for the user-depending validation, which means that our model is pretty representative and reliable. Also, we got a mean accuracy of 0.925 with a variance of 0.07 at most for the user-independing validation. Clearly we can see that our model predict the gesture's with enough certainty.

Instead, for the second domain we got worst results. For its user-depending validation we got a mean accuracy of 0.766 with a variance of around 0.06. And for the user-independing validation we got an mean accuracy of 0.59 with a high variance, 0.1 at most.

As we could not tune any hyperparameter for this model, we keep it like this, although we thought to apply some cross-validation to take the best list of templates for our recognizer, but we thought that this would not be a very good idea, as we could lose some information and it could improve one validation's score but decline the other.

Conclusion

To conclude the project, it is interesting to see the difference between the results we obtained from the edit-distance that we have implemented and the $1 recognizer. First of all, for the first domain, the two models gave an accuracy between 80% and 99% that could be interpreted as a good accuracy. It makes sense as the domain 1 is extracted from the drawn of numbers, and usually persons are more used to draw them.

For the edit distance and the user independent we can say that we have a good accuracy but with a high variance. So generalize this algorithm to a new unseen subject might give good or bad result depending on the writing of the subject and our dataset.

However, for the second domain the results of the edit-distance model seems to be quite high comparing to the $1 recognizer, some errors might be in the preprocessing of the different point which can cause these huge result for the edit distance. However, since we were unable to identify any errors, we cautiously consider the result of domain 4, with edit distance to be somewhat uncertain. As the users are required to drawn more complex figures than numbers, they might be less used to drawn them, we see a high variance and worst accuracies in both models.

Focusing on the $1 recognizer, we can check that as it works with just 2D points, it gave us pretty good results with lower variance than the baseline method for the first domain. However, for the second domain, as the figures are in 3D, sometimes just 2D points are not meaningful to represent them and this led us to worst accuracy results.

Regarding all the fact we explained before, we can say that the $1 recognizer is more reliable but it is remarkable that for complicated gesture this model could not be the best one, probably one multi-stroke recognizer could work better with domains of more complex figures.

References

[1] Jaiswal Rai Huang. “Gesture-based system for next generation natural and intuitive interfaces”. In: *Artificial Intelligence for Engineering Design, Analysis and Manufacturing* 33 (2019), pp. 54–68.