

ROB311 - TD6 - K-Means

BRAMBILLA Davide Luigi - GOMES DA SILVA Rafael

Octobre 21, 2019

The command to execute our code is: `python3 TD6.py` or `python3 TD6_reduced.py`

1 Introduction

Before starting exploring any database it is necessary to perform a pre-treatment step in it, in order to obtain only the desired characteristics to perform a certain task. In this TD, we want to classify a data-set whose correlation, *a priori*, is unknown and create a classifier that can relate an input element to a detected class.

The process of classifying unknown elements based on common characteristics, without having a classification label for the studied data is called *unsupervised learning*, and the biggest difficulty of this method is to create models that represent decision regions for new data to be classified.

1.1 The data-set

The data-set that will be treated is the data-set taken from <http://archive.ics.uci.edu/ml/data-sets.php> with the title of *Optical Recognition of Handwritten Digits Data Set* which has a set of handwritten numbers from 0 to 9, represented as gray images of 8x8 pixels size.

This data-set is composed of 5620 instances each of them with 64 attributes. This data-set is mainly used for supervised learning since each instance has also a last characteristic that is the label to which the instance belongs to.

Since our aim is to use the *k-Means algorithm* that implements a non-supervised learning, we separated the class label from the attributes and we used it only at the end in order to study the accuracy of our results.

1.2 K-Means

K-Means is an algorithm to create different *clusters* to classify data in a unsupervised way so when we don't know the labels for the instances.

The term *cluster* refers to a collection of data that are grouped together because they share some similar characteristics between them. In a *K-means* algorithm the term *k* represents the number of *clusters* that we want to create. In particular, *k* express the number of *centroids* that will be created: the goal is to associate each data to the proper *cluster* while keeping the *centroids* in the center of the *clusters*. In our example we would like to recognize the 10 digits so we will have to create 10 classes.

Actually, in our algorithm, we have seen that we could have associated two different classes to the 9 digit because there were mainly two 9 with different characteristics. This brought us, in a second part of the algorithm, to have 11 *clusters* and then, at the end, unify the tenth and the eleventh class giving associating them to the 9 digit.

1.3 The algorithm t-SNE

In order to visualize our results with a *scatter graph* we had to reduce the dimension of our data-set from 64 to 2 in a way to have the possibility to represent a bi-dimensional data-set.

In order to do this, we used the *t-distributed stochastic neighbor embedding algorithm (t-SNE)*.

This algorithm uses a nonlinear technique in order to reduce a high-dimensional data to a low-dimensional ($2D$ or $3D$) point in a way that similar objects will have the same position in a $2D$ space.

This algorithm is composed by 2 main steps:

- the construction of a probability distribution that permits to have an high possibility that similar objects grouped together and a low possibility that different objects are grouped together.
- the definition of probability distributions over the points in the low-dimensional space. Then the algorithm minimize the *Kullback-Leibler divergence* between the two distributions with respect to the locations of the points in the map.

2 Results

In order to analyze the algorithm developed, we performed measurements of accuracy for the results using *scikit-learn* tools. Also, the algorithm developed contains the following main stages:

- **Data load:** Reads the data and save all the information into arrays.
- **Data reduction (In a second moment):** This stage will be used in order to better represent the dataset and visualize the results
- **Defining the amount of clusters to divide the data:** In this case, the minimal number of clusters is 10 (since our dataset represents numbers from 0 to 9). But as we will see, there might be necessary to use more than 10 classes
- **Data fitting using k-means:** Uses the k-means method contained in the sciklearn library to perform the k-means in the dataset
- **Data prediction:** Uses the predictor created in the last stage to predict the values of the dataset
- **Centroids identification:** The k-means method from the sciklearn library initializes the centroids for all the clusters in a random way. Thus, in order to use the labels of the dataset to verify the results of the algorithm, it is necessary to verify to what class each cluster detected belongs
- **Evaluation of the results:** To evaluate the performance of our algorithm, we compared the labels given by the prediction stage with the labels of the data. In this case we used different methods of evaluation in order to measure local and global efficacy

In this section we will present the results obtained with both original dataset and data set after dimension reduction.

2.1 Measurement tools

The tools that we've used are:

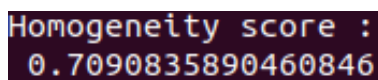
- the **Homogeneity Score:** that is a metric that measures the homogeneity of a cluster labeling given a ground truth. A clustering result satisfies homogeneity if all of its clusters contain only data points which are members of a single class.
- the **F1 Score:** that can be interpreted as a weighted average of the precision and recall: it takes a value between 0 and 1.
 - the *precision* is fraction of relevant instances among the retrieved instances
 - the *recall* is the fraction of the total amount of relevant instances that were actually retrieved.
- the **Overall accuracy** where different parameters like *precision*, *recall*, *f1-score* are reported in a table.
- the **Confusion Matrix** is the square matrix that contains on its main diagonal the number of instances that are correctly classified and in the other positions the instances that are classified to a certain class wrongly.

2.2 Results for full-dimension data

In order to test this code we have to run: `python3 TD6.py`.

2.2.1 Homogeneity Score

The homogeneity score is 0.7091(70.91%). This is not a very high result and it is due to the fact the recognition of the 9 digit has an high error rate.



```
Homogeneity score :
0.7090835890460846
```

Figure 1: Overall homogeneity for the full-dimension data-set with $k = 10$

2.2.2 F1 Score

For the tests in the full dataset we could realize that for the numbers "9" and "1" the values for the accuracy of the matches weren't very good. As already remarked, the digit "9" is the one that has the lowest score and in order to try to solve this problem, we tried to change the number of classes for the classification with the k-means algorithm. The table below show the result obtained for a k-means of 10, 11 and 12 classes.

Value of k	Class 0	Class 1	Class 2	Class 3	Class 4	Class 5	Class 6	Class 7	Class 8	Class 9
10	0.9887	0.4769	0.8595	0.5903	0.8753	0.8941	0.9670	0.8434	0.5708	0.1850
11	0.9887	0.7482	0.8604	0.8933	0.9244	0.8605	0.9695	0.8997	0.7928	0.7493
12	0.9887	0.8071	0.8571	0.8837	0.8685	0.8667	0.9695	0.8545	0.7927	0.7056

Table 1: F1 score for every class for the full-dimension data-set

2.2.3 Accuracy Result

```

ACCURACY :
          precision    recall  f1-score   support

     0:       0.99      0.99      0.99        178
     1:       0.79      0.34      0.48        182
     2:       0.87      0.85      0.86        177
     3:       0.44      0.90      0.59        183
     4:       0.97      0.80      0.88        181
     5:       0.96      0.84      0.89        182
     6:       0.96      0.97      0.97        181
     7:       0.92      0.78      0.84        179
     8:       0.48      0.70      0.57        174
     9:       0.26      0.14      0.19        180

 accuracy                   0.73        1797
 macro avg       0.76      0.73      0.73        1797
 weighted avg    0.77      0.73      0.72        1797

```

Figure 2: F1 Score for every class for the full-dimension data-set with k = 10

2.2.4 Confusion Matrix

```

Confusion matrix:
[[176  0  0  0  2  0  0  0  0  0]
 [  0 62 21  1  0  0  4  0 94  0]
 [  1  2 150  8  0  0  0  3 13  0]
 [  0  0  0 165  0  1  0  7  8  2]
 [  0  4  0  0 144  0  0  1  7 25]
 [  0  0  0 26  1 152  1  0  0  2]
 [  1  1  0  0  1  0 176  0  2  0]
 [  0  0  0  0  0  0  0 140  3 36]
 [  0  6  1 31  0  2  2  1 121 10]
 [  0  3  0 145  0  3  0  1  2 26]]

```

Figure 3: Confusion Matrix for the full-dimension data-set with k = 10

2.2.5 Conclusion

As we can see, as we increase the amount of classes of the k-means algorithm, the accuracy for each classes increases too. However, since we are working with a data-set of 64 dimensions, it is hard to visually see the reason why it was necessary to increase the number of classes for the k-means algorithm. In order to better understand what was the problem and also to plot the results for the data-set, we reduced the size of the dimensions of the data-set to 2 dimensions using the *t-SNE* algorithm, as it will be shown in the next section.

2.3 Results for reduced-dimension data

In order to test this code we have to run: `python3 TD6-reduced.py`.

The initial problem with the data-set in terms of efficiency was due the amount of characteristic of the elements in the database. To try to solve this problem, the *HOG*, *PCA*, were verified before choosing the *t-SNE*. However, due to the complexity in the image principal characteristics, maybe due to the small resolution, the *HOG* and *PCA* methods resulted non-effective.

The Figure 4 shows the data after being separated using the *t-SNE* algorithm. It is possible to verify that the *t-SNE* worked really well separating the data.set in groups according to the characteristics in common.

However, it is important to notice also that for some classes, the data seems to present two different regions grouped together (such as the class "1" and the class "9"). This was one of the problems why the algorithm run before presented a low accuracy for those numbers, and a solution for this problem is to increase the amount of classes that the k-means will be dealing with.

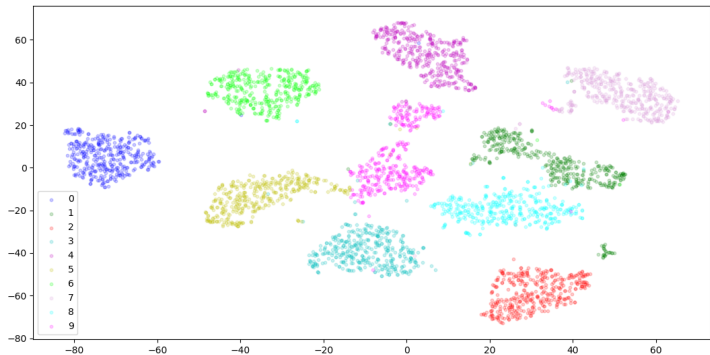


Figure 4: Data-set visualization after the dimension-reduction

Below, we will present the results in the case we will use two classes for the "9" digit (so using 11 clusters) in order to take into account the two regions that effectively represent the same number.

2.4 Test with 11 classes

The Figure 5 shows the Figure 4 after being processed by the *k-means* algorithm. As we can see, the image shows already the 11 *centroid*, placed in the center of each cluster, and we can notice that the *k-means* algorithm detects that the class "9" needs another cluster since it is the more sparse class.

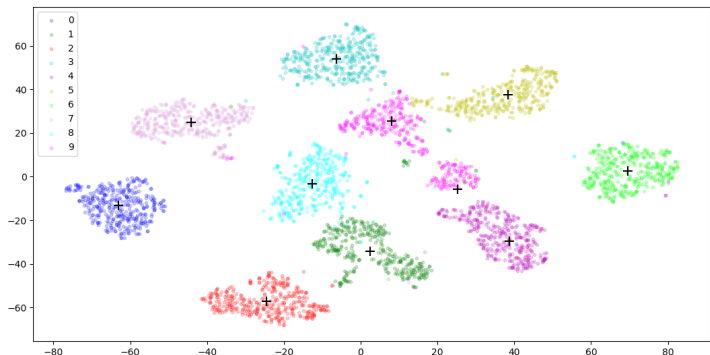


Figure 5: Data set visualization after the dimension-reduction, with 11 classes

2.4.1 Homogeneity Score

The result for the Homogeneity measure becomes now 0.92633 (92.63%). As we can see, the result is highly improved with respect to the full-dimension data-set with 10 clusters.

HOMOGENEITY SCORE:
0.9263372867532997

Figure 6: Homogeneity Score for the reduced-dimension data-set using 11 classes

2.4.2 F1 Score

Class 0	Class 1	Class 2	Class 3	Class 4	Class 5	Class 6	Class 7	Class 8	Class 9
0.9973	0.9475	0.9743	0.9844	0.9164	0.9556	0.9894	0.982	0.9828	0.8763

Table 2: F1 score for every class for the full-dimension data-set

2.4.3 Accuracy Results

ACCURACY :				
	precision	recall	f1-score	support
0	1.00	0.99	1.00	376
1	0.97	0.93	0.95	389
2	0.95	1.00	0.97	380
3	0.99	0.98	0.98	389
4	0.99	0.85	0.92	387
5	1.00	0.92	0.96	376
6	0.98	0.99	0.99	377
7	0.98	0.99	0.98	387
8	0.99	0.98	0.98	380
9	0.80	0.97	0.88	382
accuracy			0.96	3823
macro avg	0.96	0.96	0.96	3823
weighted avg	0.96	0.96	0.96	3823

Figure 7: Accuracy for the reduced-dimension data-set using 11 classes

2.4.4 Confusion Matrix

CONFUSION MATRIX :										
[[374	0	0	0	1	0	1	0	0]
[0	361	20	0	0	0	0	1	2	5]
[0	0	380	0	0	0	0	0	0	0]
[0	3	0	380	0	1	0	1	1	3]
[0	0	0	0	329	0	3	1	0	54]
[0	1	0	0	0	345	0	0	0	30]
[0	2	0	0	0	0	375	0	0	0]
[0	1	0	1	1	0	0	382	0	2]
[0	5	0	0	0	0	2	0	372	1]
[0	0	0	2	0	0	0	6	2	372]]

Figure 8: Confusion Matrix for the reduced-dimension data-set using 11 classes

3 Conclusion

With this TD it was possible to verify the efficacy of the *k-means* algorithm when dealing with the classification of data using unsupervised learning, as the results obtained for the overall accuracy for the results were over 0.7. However, some elements showed a lower accuracy due to the fact that for some classes (such as the number 1 and 9), there are major differences in the characteristics probably due to the different way of writing the same number.

Thus, in order to better understand the data set and also to improve the performance of the *algorithm k-means*, a reduction of dimension using the *t-SNE algorithm* was proposed. Besides enhancing the accuracy of the data, reducing the dimensions of the data-set allowed us to *visualize* the results of the *clusters* obtained with the k-means method, which would be impossible if we were working with the full data-set.