# Algorithm description

In this section I described algorithm to extract fragments which are necessary for implementation because of matrix operations are essentially faster than for loops both in Matlab and python.

## Data

We have -dimensional dataset with labelled data: for each input vector we have label . We have also unlabelled dataset with elements . Let us denote labels as

## Problem

We want to find -dimensional subspace such that for projection of points of dataset the distance between projections of points of different classes will be greater than distance between projections of points of one class.

We also want to have projection of subset as close as possible to projection of subset .

## Model

According to (6) and (7) in [1] for weights matrix ( are numbers of observations) we need to find the greatest eigenvalues of matrix with elements

|  |  |
| --- | --- |
|  | (1) |

where is one data point.

In formula (1) the second summand can be rewritten as product of three matrix

Check of myself.

The first summand can be rewritten as sum of product of elements of three vectors , where vector is sum of column elements of matrix :

## Additional calculations

For our algorithm we also need to find for each element of dataset nearest neighbours from dataset . Let us denote this set and element of this set as .

Let us denote normalised distance (squared distance?) between point and as

Weight of nearest neighbour can be constant or normalised distance. Really we can consider arbitrary set of weights .

## Weights for DAPCA

We have two datasets and . We have unique different labels in dataset . For each element of dataset we have set of NN in dataset . Now we can define weights matrix:

1. If then (attraction of projections of points with the same label in labelled dataset).
2. If then (repulsion of projections of points with different labels).
3. If then (repulsion of projections of points of unlabelled dataset).
4. If then (attraction between projection of unlabelled point and nearest labelled neighbour).
5. If then (no action!).

## Scheme with normalised weights

We have two datasets and with and objects correspondingly. We have unique different labels in dataset . Number of cases with label is . For each element of dataset we have set of NN in dataset . Now we can define weights matrix:

1. If then (attraction of projections of points with the same label in labelled dataset).
2. If then (repulsion of projections of points with different labels).
3. If then (repulsion of projections of points of unlabelled dataset).
4. If then (attraction between projection of unlabelled point and nearest labelled neighbour).
5. If then (no action!).

## Parameters of algorithm

is user defined. Mandatory. We do not define any default value for it.

is optional. Default value is 1.

is optional. Default value is 0.5. I am not sure.

is optional. Default value will be defined later.

is optional. Default value will be defined later.

is optional. Default value is 1 for all pairs of labels.

is optional. Default function is constant 1.

## Iterative DAPCA

Iterative DAPCA firstly calculated kNN for each point of set . Then form matrix (1). Calculated projections onto the first PComs. Recalculate kNN in space of projections. Recalculate matrix Q. Repeat until convergence or maximal number of iterations.

Algorithm:

1. Coordinates for kNN XkNN = X, YkNN = Y. Set list of kNN to zeros. IterNum = 0;
2. Infinite loop
   1. OldkNN = kNN
   2. Calculate kNN in XkNN for each element of YkNN.
   3. If for each then stop
   4. Calculate matrix through (1).
   5. Calculate the first eigenvectors () which corresponds to the first greatest eigenvalues (not to the eigenvalues with greatest absolute value of eigenvalue).
   6. Calculate projections of and to hyperplane of the first PComs: , .
   7. IterNum = IterNum + 1
   8. If IterNum = maxIter then stop
   9. End of loop (repeat loop body from item 2.1)
3. Return

## Matrix calculation

If we consider formula (1) we can see that matrix contains several summands with respect to data sets and .

For all iteration of iterative DAPCA the only changed part is matrix . Moreover, elements of matrix which related to either dataset or only are constant too. As a result the only changed part of matrix is part of interactions of datasets and .

Let us consider matrix :

Matrices and are iteration independent and depends on itearation through nearest neighbours.

Matrix can be represented as

Let us consider matrix :

As a result we have two summand which can be calculated once and one summand which must be calculated on each iteration.

Each element of matrix must be calculated on each iteration because of each element of vector depends on elements of submatrix or :

To avoid of repeated calculations we can calculate ones the two sub sums and then calculate only inconstant part:

|  |  |
| --- | --- |
|  | (2) |
|  | (3) |

Usage of once calculated summands allows us not only avoid of additional calculations but also allow us to avoid of saving or reforming of matrix . Matrix can have huge size and can create problems.

## Further decomposition of calculations

Matrix is uniformly calculated (there is no deeper structure) since the matrix can be further decomposed with respect to classes. Let us consider dataset with classes. In this case we can decompose matrix to the following form

The most useful property of all matrices that this matrix are constant:

|  |  |
| --- | --- |
|  | (4) |
|  | (5) |

Moreover, we can directly calculate submatrix of matrix . Firstly we need to decompose matrix into submatrices:

Now we can calculate

Let us calculate one of summand. We know that matrix is matrix of ones multiplied by constant (4) or (5). Let us denote this constant and element of attribute of object of matrix . We can find

Let us calculate

where is sum of attribute (column) values of matrix :

|  |  |
| --- | --- |
|  | (6) |

Now we can complete calculation:

Finally we can write

|  |  |
| --- | --- |
|  | (7) |

For diagonal element we can write

|  |  |
| --- | --- |
|  | (8) |

For matrix we also have property . As a result we can write

|  |  |
| --- | --- |
|  | (9) |

## Calculation of

To calculate constant parts of vector in (2) and (3) we should to calculate sum of row elements of matrices and . There is no any problem with matrix because of all elements of matrix are equal to . As a result we can write

|  |  |
| --- | --- |
|  | (10) |

Elements of matrix have constant fragments one for each class. For class we have:

|  |  |
| --- | --- |
|  | (11) |

To produce more uniform calculation we can use the following matrix :

|  |  |
| --- | --- |
|  | (12) |

In this case we can write (11) for class as

|  |  |
| --- | --- |
|  | (13) |

# Debugging

## Reproduction of SPCA

Reproduction of SPCA is presented in Figure 1. As we can see figures are not exactly the same because of different normalisation of repulsion. However we can see qualitatively the same figures.

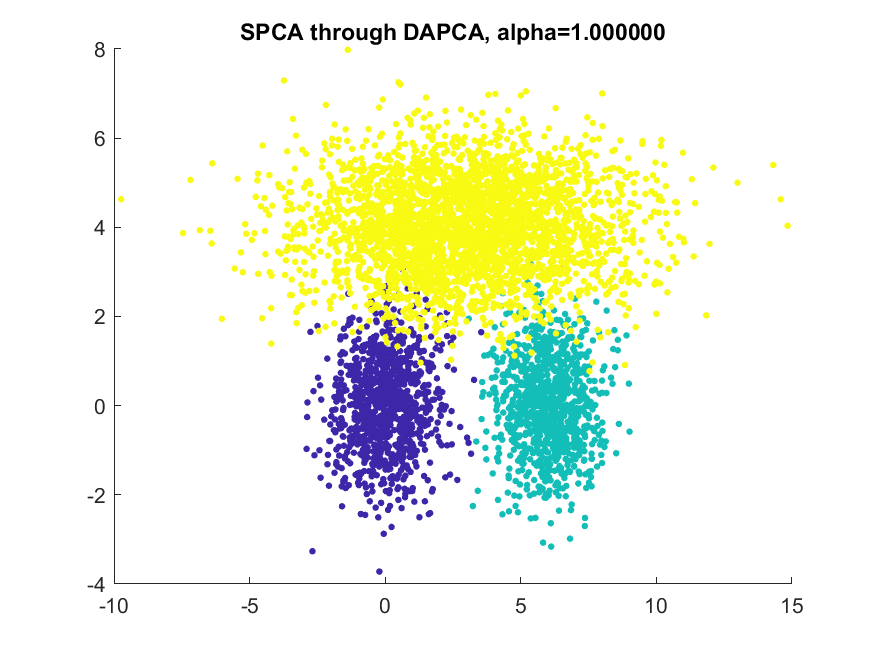
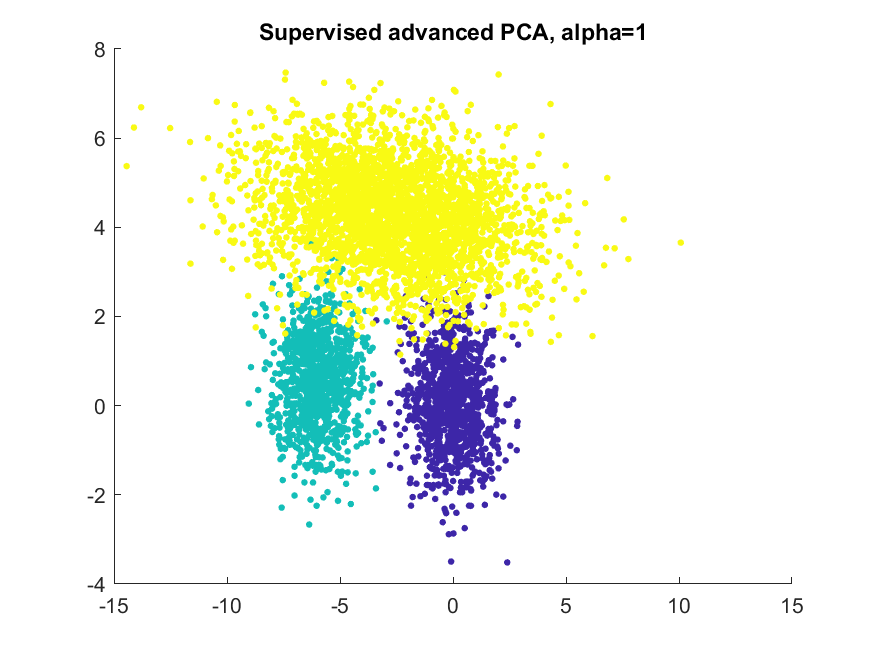


Figure 1. The SPCA (left) and DAPCA (right) for the same dataset

## Variation of repulsion

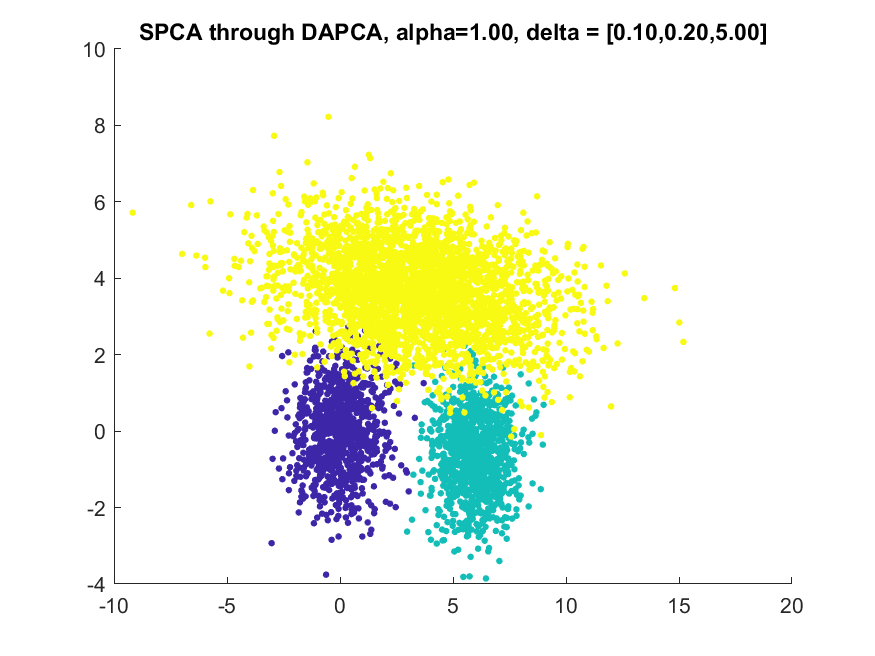
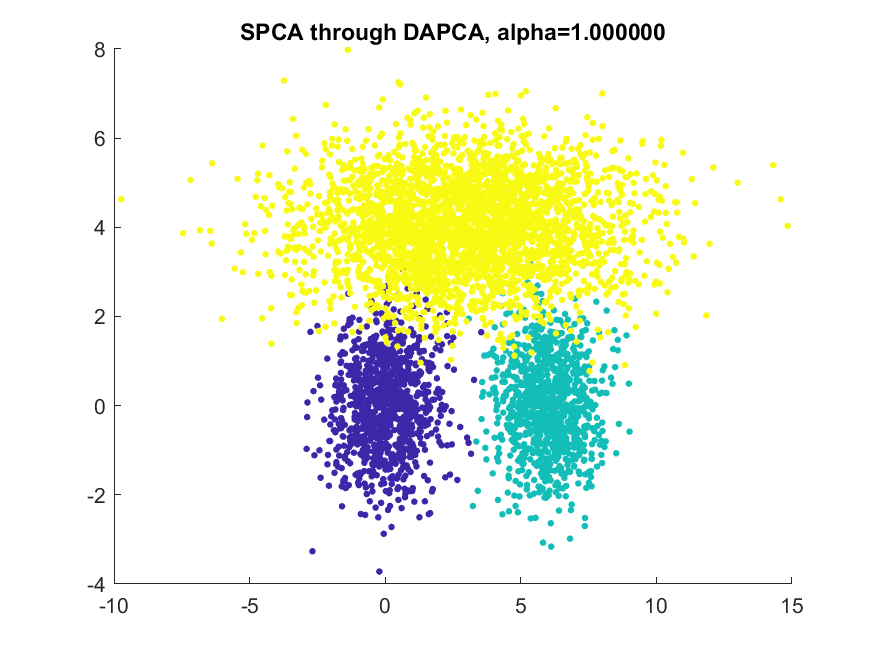


Figure 2. DAPCA with delta variation

We can see difference in clouds but it is necessary to understand how to use this difference and what does it mean. I think that we should calculate weighted accuracies of Fisher’s discriminant (for example) with weights of error defined by delta. To discuss.

## Repulsion in Y

Since Y was very close to X (see Figure 3) I added shift (-5, -5) to Y.

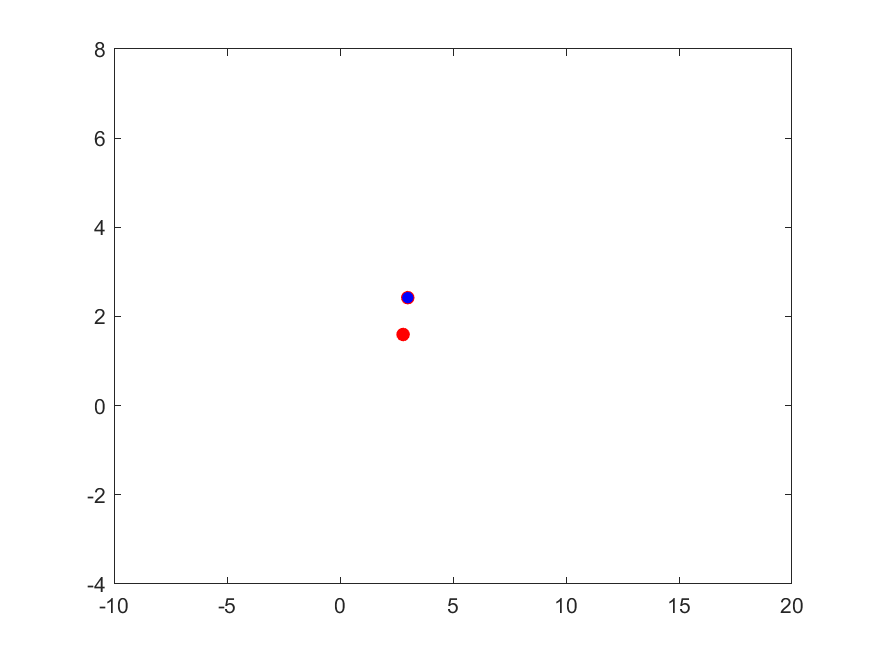
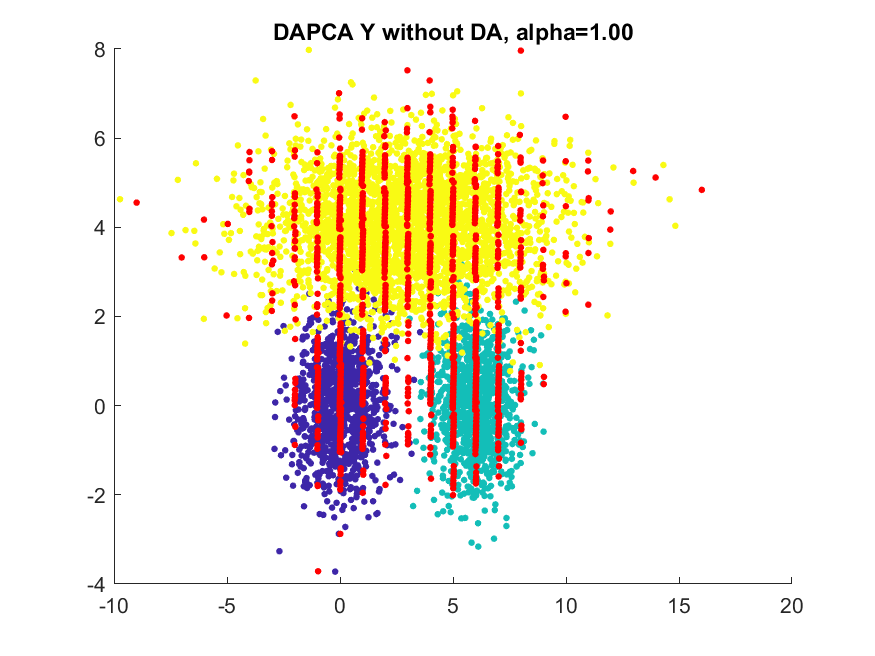


Figure 3. X and Y (red) in SPCA

For shifted Y we have following figure (Figure 4)

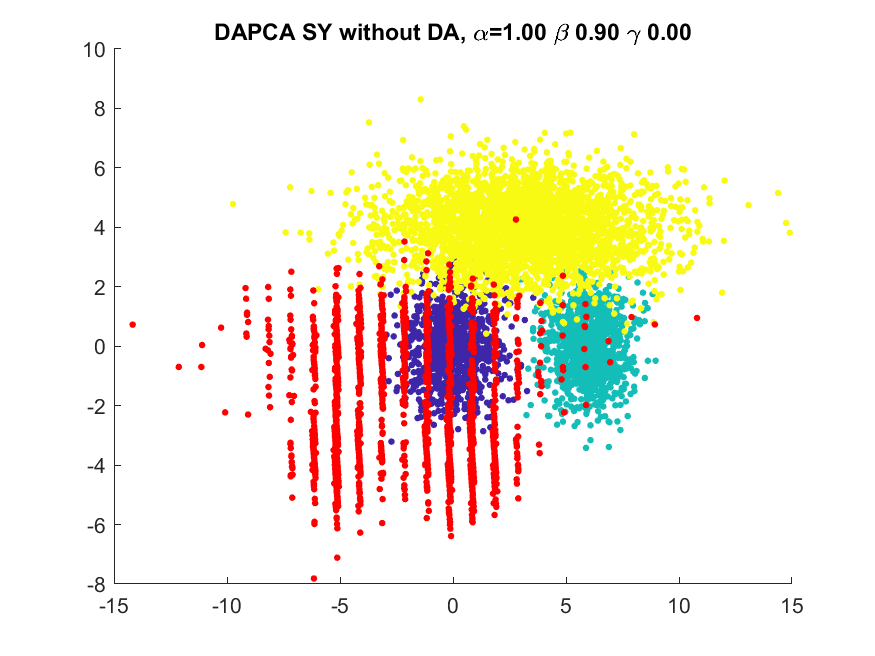
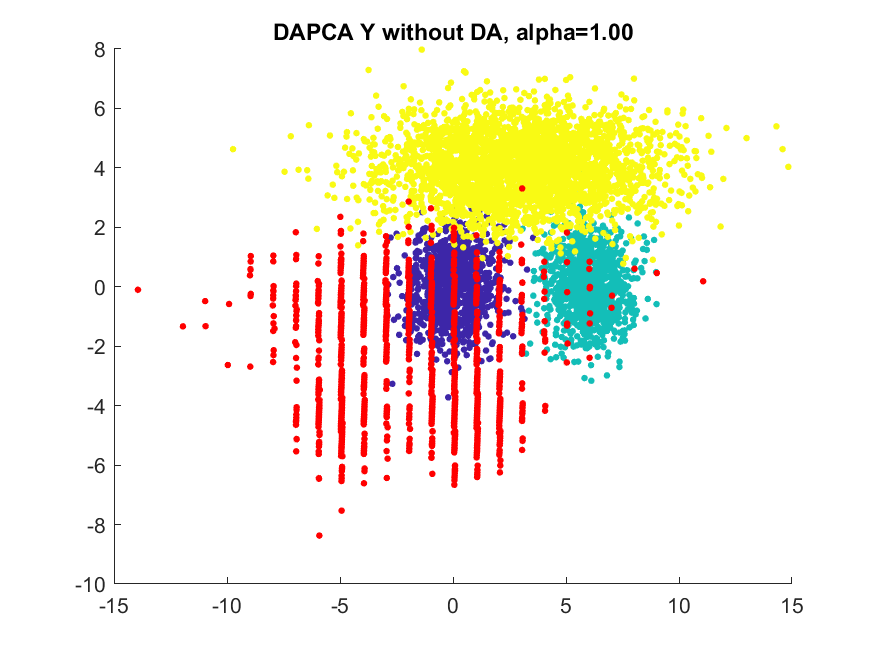


Figure 4. X and shifted Y without Y in PCom search (left) and with Y repulsion (right)

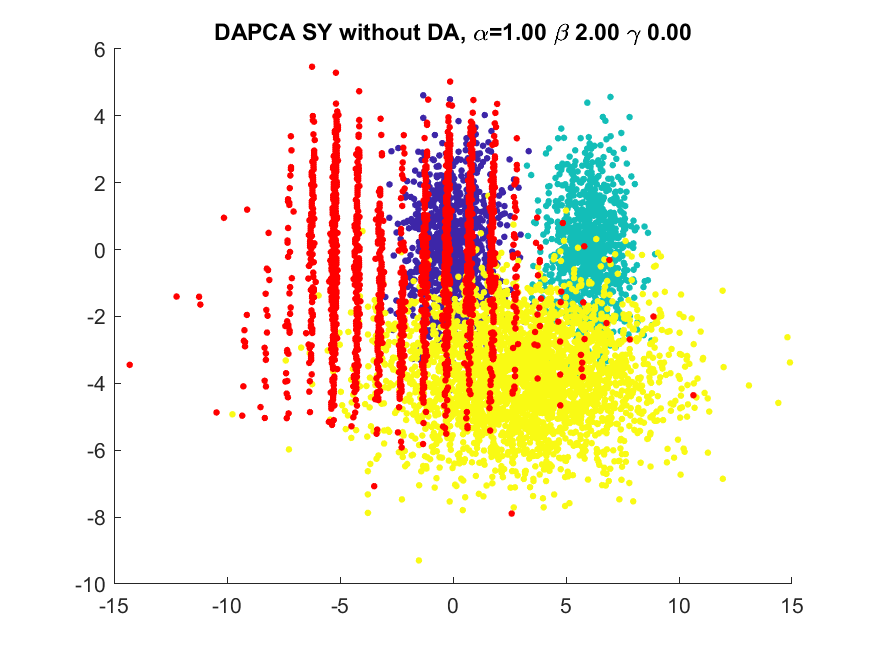
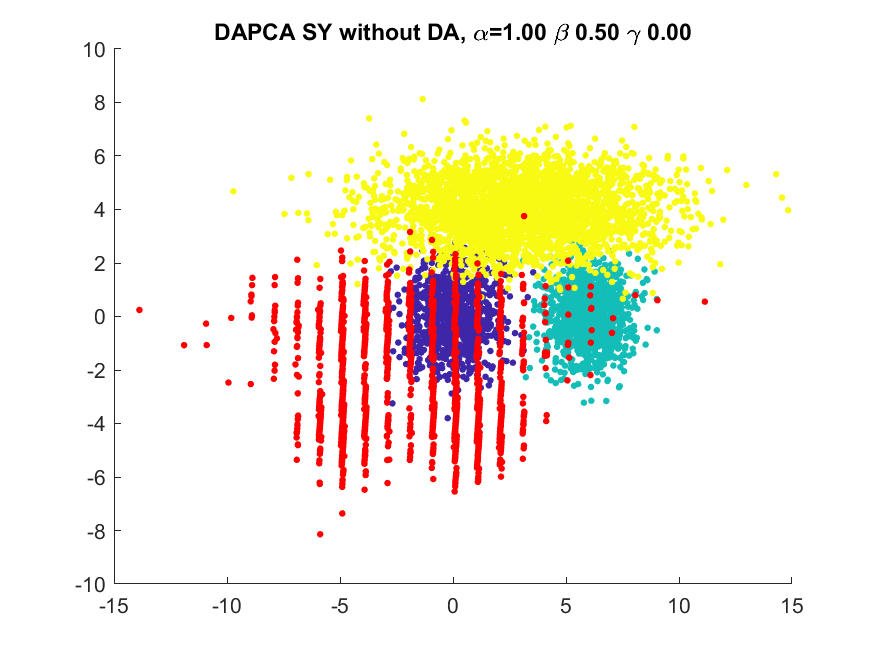


Figure 5. Influence of Y repulsion (beta) for Y repulsion only

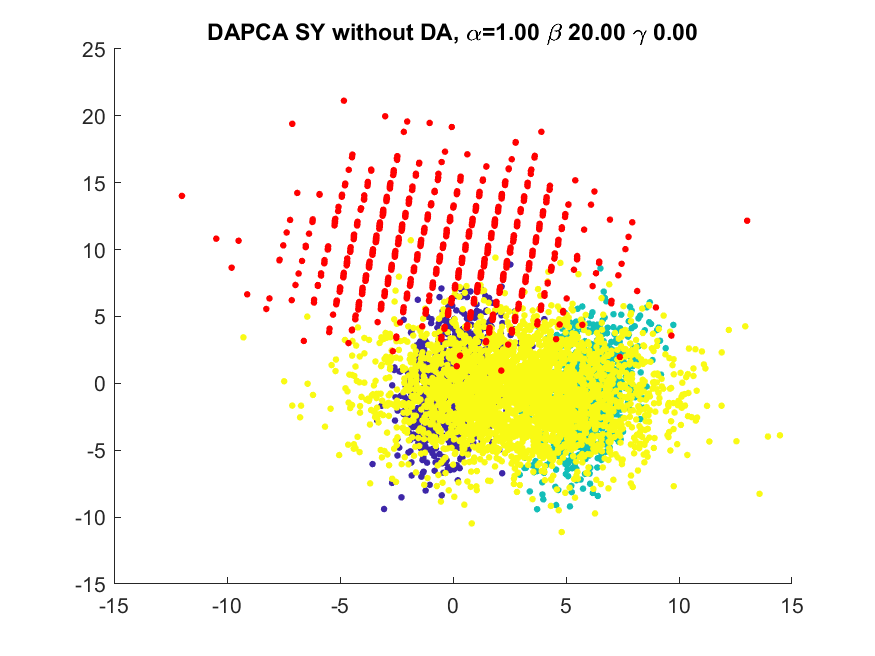
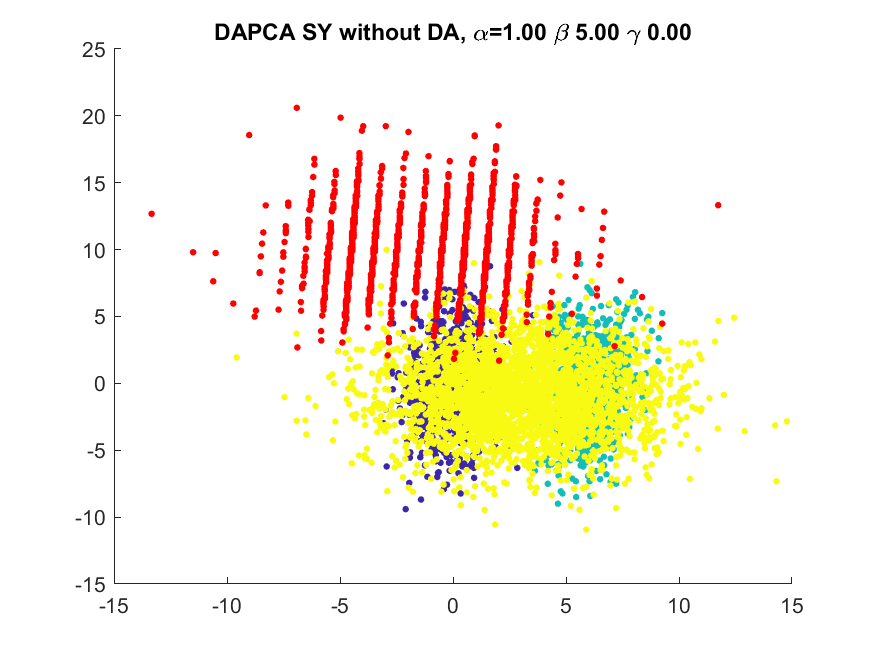


Figure 6. Influence of Y repulsion (beta) for Y repulsion only

## Attraction of Y to X

For this study we fixed and variate and number of NN.

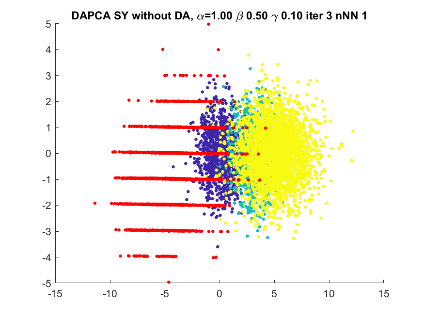
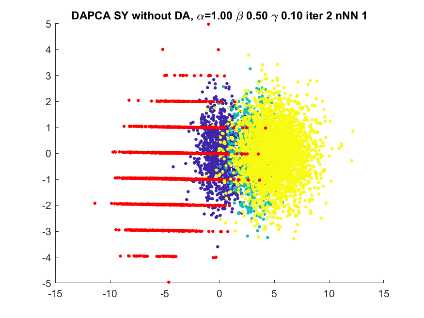
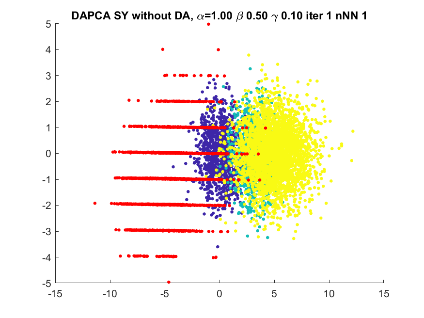


Figure 7. Changes results iteration to iteration

Figure 7 shows almost the same images but there are some differences. To check differences we calculated angles between PComs and distances between centroids of X and Y. Results are presented in Table 1. We can see that almost nothing changes.

Table 1. Comparison of three sets of PComs

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | PCom 1 | | | PCom 2 | | |  |
|  | Iter 1 | Iter 2 | Iter 3 | Iter 1 | Iter 2 | Iter 3 |  |
| Iter 1 | 0 | 0.005692 | 0.005678 | 0 | 0.003335 | 0.003341 | 76.60952 |
| Iter 2 | 0.005692 | 0 | 0.000112 | 0.003335 | 0 | 0.000007 | 76.60592 |
| Iter 3 | 0.005678 | 0.000112 | 0 | 0.003341 | 0.000007 | 0 | 76.60622 |

We also can see huge differences between and . For these two cases we have angle between the first PComs 60 degrees and between the second PComs 89 degrees. The distance between centroids of and is 53.15 which is less than 76.61. It is not clear which case is closer to adapted domain. For we have very good domain adaptation but the supervised effect is destroyed. Distance between centroids of and is 2.35 and it is really very good value. It is interesting that increasing of can destroy quality of domain adaptation.

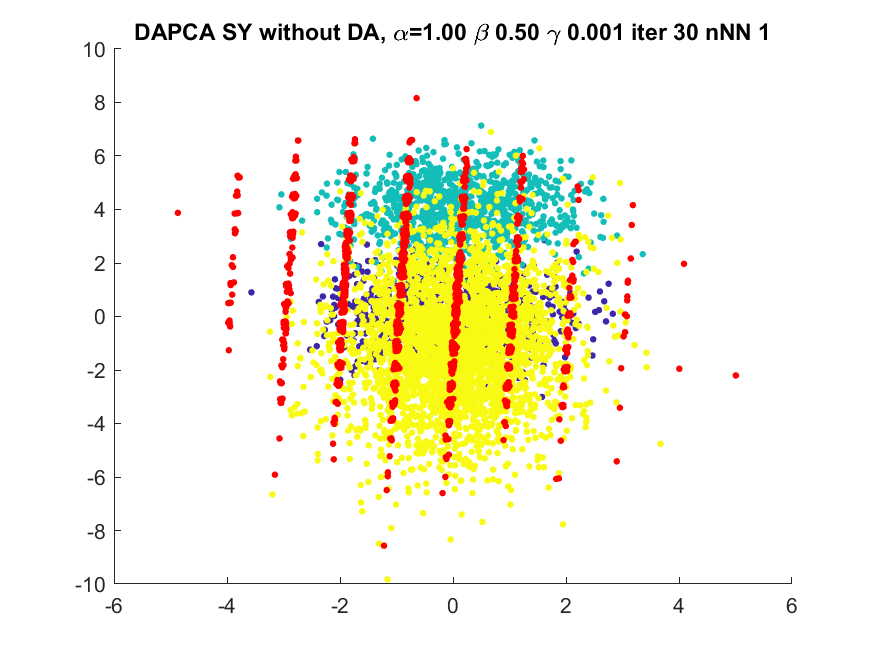


Figure 8. The best found domain adaptation

# Bibliography

1. Gorban, A.N., Grechuk, B., Mirkes, E.M., Stasenko, S.V. and Tyukin, I.Y., 2021. High-dimensional separability for one-and few-shot learning. *arXiv preprint arXiv:2106.15416*.