/\* Principal Component Analysis of the data: using correlation matrix.

data - input matrix;

height, width - sizes of the input;

numOfPrincipalComponents - number of principal components to be computed;

principalComponents - the principal components computed;

approxData - approximation of the input data using principal components obtained. \*/

int pca::pcaUsingCorrelationMatrix(Float \* data,

const int height,

const int width,

const int numOfPrincipalComponents,

Float \* principalComponents,

Float \* approxData) noexcept

{

// check number of principal components

if (height < numOfPrincipalComponents)

return 1;

// assign data and Eigen matrix

Map< Matrix<Float, Dynamic, Dynamic, RowMajor> > dataMatrix(data, height, width);

//cout << "\nD:\n" << dataMatrix << endl;

Vector<Float, Dynamic> means = dataMatrix.rowwise().mean();

//cout << "\nmeans:\n" << means << endl;

Matrix<Float, Dynamic, Dynamic> corMatrix

= dataMatrix \* dataMatrix.transpose() / width - means \* means.transpose();

// The following solver computes eigen vals & vectors: the order of eigen vals is increasing.

SelfAdjointEigenSolver<Matrix<Float, Dynamic, Dynamic>> eigensolver(corMatrix);

// Check result of eigen values & vectors computation.

if (eigensolver.info() != Success)

return 2;

// get feature vectors, taking into account increasing order of computed eigen values

Matrix<Float, Dynamic, Dynamic, ColMajor> featureVectors

= (eigensolver.eigenvectors().rowwise().reverse())(all, seq(0, numOfPrincipalComponents - 1));

// assign principal components and Eigen matrix

Map< Matrix<Float, Dynamic, Dynamic, RowMajor> > princCompMatrix(principalComponents, numOfPrincipalComponents, width);

// compute principal componets

princCompMatrix = featureVectors.transpose() \* (dataMatrix.colwise() - means);

// computation of approximation

if (approxData != NULL)

{

// assign data and Eigen matrix

Map< Matrix<Float, Dynamic, Dynamic, RowMajor> > approxMatrix(approxData, height, width);

approxMatrix = (featureVectors \* princCompMatrix).colwise() + means;

}

return 0;

}