PRNN Assignment - 3

1. Question 1

Here we have to implement decision tree and note accuracy.

1.1. Implementation

- At first datasets for classification problem 1 (P3 of Assignment 1) and classification problem 2 (P5 of Assignment 1) is loaded into colab and datas are separated into train and test sets.
- Then below mentioned functions are created that are needed to create decision trees.
- CLASS NODE: It is a class that stores datas related to a particular node in a tree like for instance if the node is a leaf or not, its depth, feature on which it separates etc.
- ENTROPY: It takes in a dataset and calculates its entropy and returns the entropy value.
- GINI_INDEX: It takes in a dataset and returns the gini index value of the dataset.
- GROW_TREES_ENTROPY: It takes a dataset,current depth and maximum depth and creates tree recursively until maximum depth is reached. It creates trees on basis of that value that minimizes entropy. It returns back root node of the tree created.
- GROW_TREES_GINI: It takes a dataset,current depth and maximum depth and creates tree recursively until maximum depth is reached. It creates trees on basis of that value that minimizes gini index. It returns back root node of the tree created.
- RUN_DECISION_TREE: It takes dataset and root of the node and returns back the predicted values according to the tree pointed by root node.
- Some other functions required like for calculating maximum occuring class in a dataset are also created.
- The decision tree is first created on train dataset and tested on test dataset for various depths

1.2. Results and Observations

The accuracy for the above mentione two classification problems are found for 10 depth trees and 20 depth trees for gini index and entropy each. The table below shows the results obtained

Dataset	Impurity	Depth	Train Acc	Test Acc
D1	Gini	10	56.39	54.24
D1	Gini	20	74.45	50.88
D1	Entropy	10	56.48	55.26
D1	Entropy	20	79.38	48.90
D2	Gini	10	89.35	90.22
D2	Gini	20	96.93	91.99
D2	Entropy	10	89.58	90.71
D1	Entropy	20	97.48	92.19

From the table we can observe as the depth of tree increases the train accuracy increases as the decision tree overfits the data but the test accuracy decreases with increase in depth. The decrease in test accuracy is not that pronounced in dataset 2. To counter this effect of decision tree we use bagging (random forest), that removes this problematic nature of decision tree and gives good accuracy in both train and test datast.

Precision, recall, confusion matrix and F1 scores are found for each classes, each model and each dataset. For F1 score, precision and recall refer colab notebooks. precision, recall and F scores for dataset D1 are between 0.85 and 0.95 for all models and between 0.45 to 0.55 for D2 datasets and all models. For exact values please refer colab notebook.

1.3. Problems Faced

Here one of the problem faced is the datas of different questions are aligned differently, for instance for dataset 1 the classes are in last column but for dataset 2 classes are in first column. Hence they are first converted to a common format and then the above algorithm is applied.

The second problem is the step size to take while finding the best decisions on a particular feature, to reduce the steps size of checking the datasets are normalised, so that each feature lies between 0 and 1 and then a step size of 0.01 is taken over all features.

2. Question 2

Here our aim is to implement random forest on classification problems of Assignment 1.

2.1. Implementation

- Here the datasets for classification problem 1 (P3 of Assignment 1 - D1) and classification problem 2 (P5 of Assignment 1 - D2) is loaded into colab and datas are separated into train and test sets
- The functions of previous questions are loaded as we also need to create trees here.
- Extra functions are added that are required by the random forest.
- GROW_TREE_WITH_SLCT_FTR: It takes a dataset,current depth,maximum depth and a list of features to select from and creates tree recursively until maximum depth is reached. It creates trees on basis of that value that minimizes entropy. It returns back root node of the tree created.
- RANDOM_FOREST: It takes dataset, number of features(to select for each tree) and number of trees in the forest as input and returns back a list of nodes of the trees created.
- RUN_RANDOM_FOREST: It takes dataset and list of roots of trees of the random forest and returns the predicted value of dataset.
- The random forest is created for varying trees and features and output shown.

2.2. Results and Observations

The algorithm is run for a fixed depth of 10 nodes and various features and trees and results are tabulated as below.

Dataset	Features	Trees	Train Acc	Test Acc
D1	5	8	51.29	48.92
D2	5	8	88.26	90.39

F1 score for D2 case:
[0.903, 0.875, 0.934, 0.909, 0.931, 0.920, 0.873, 0.808, 0.932, 0.944]
Precision for D2 case:
[0.906, 0.858, 0.922, 0.875, 0.916, 0.934, 0.818, 0.911, 0.938, 0.979]
Recall for D2 case:
[0.899, 0.892, 0.946, 0.946, 0.947, 0.907, 0.935, 0.727, 0.925, 0.913]

F1 score for D1 case:
[0.508, 0.524, 0.492, 0.496, 0.414]
Precision for D1 case:
[0.430, 0.511, 0.525, 0.534, 0.477]
Recall for D1 case:
[0.620, 0.537, 0.462, 0.464, 0.366]

To understand how the accuracy vary with number of trees and features, graphs are also plotted. Here the graph is shown only for D2 dataset case, simmilar graph is obtained for D1 case also, refer to colab notebook for more graphs. Confusion matrix is also attached for D1 and D2 case.

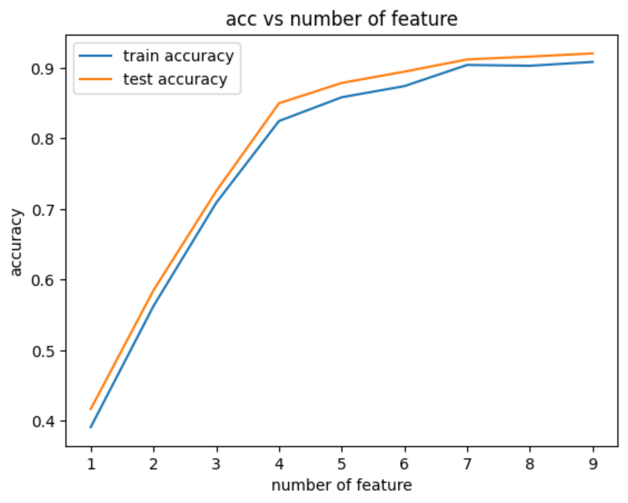


Figure 1. acc vs number of features

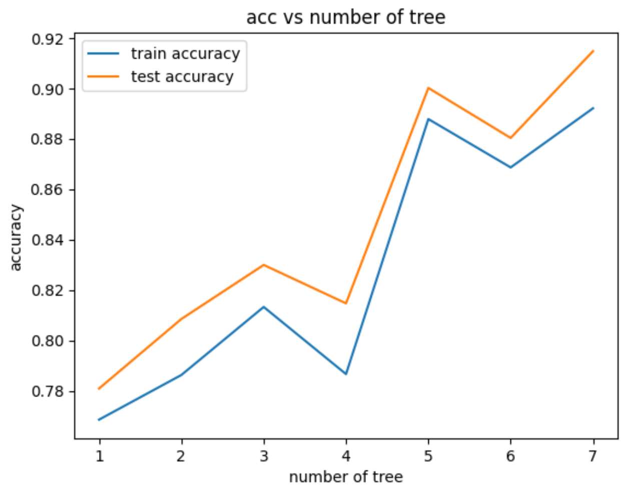
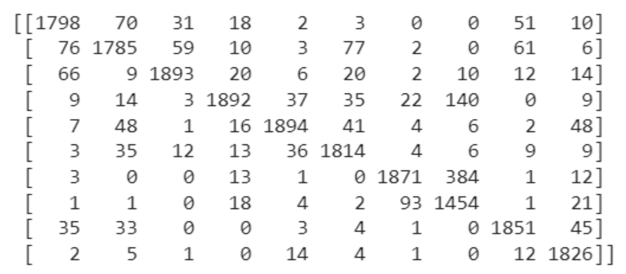


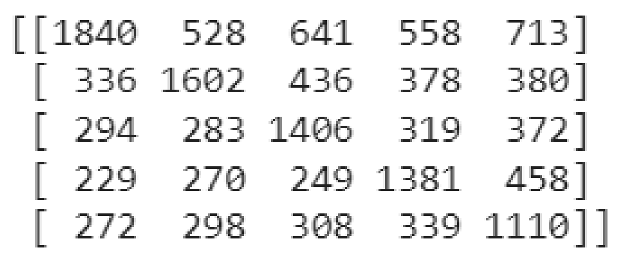
Figure 2. acc vs number of tree

Accuracy should increase with increase in number of trees, as we have more number of datasets on which the model is trained and hence better generalizes the model. In our graph(Fig 2) also accuracy is increasing with increase in number of trees, only upto 8 number of trees are considered



```
[[1798   70   31   18    2    3    0    0   51   10]
 [  76 1785   59   10    3   77    2    0   61    6]
 [  66    9 1893   20    6   20    2   10   12   14]
 [   9   14    3 1892   37   35   22  140    0    9]
 [   7   48    1   16 1894   41    4    6    2   48]
 [   3   35   12   13   36 1814    4    6    9    9]
 [   3    0    0   13    1    0 1871  384    1   12]
 [   1    1    0   18    4    2   93 1454    1   21]
 [  35   33    0    0    3    4    1    0 1851   45]
 [   2    5    1    0   14    4    1    0   12 1826]]
```

Figure 3. Confusion matrix for D2



```
[[1840  528  641  558  713]
 [ 336 1602  436  378  380]
 [ 294  283 1406  319  372]
 [ 229  270  249 1381  458]
 [ 272  298  308  339 1110]]
```

Figure 4. Confusion matrix for D1

because of time limit.
Accuracy should increase with increase in number of features and it is what happening as shown in fig 1.

2.3. Problems faced

Dataset D1 and D2 have different formats (like class column numbers are different) and hence has to be brought to a common format to run the algorithm.

3. Question 3

Here we have to implement the adaboost algorithm.

3.1. Implementation

- Here the datasets for classification problem(PCA - MNIST) is loaded into colab and datas are separated into train and test sets
- Neural net functions created in assignment 2 are loaded here.
- Tree functions created in the previous questions are also loaded here.
- Different classifier functions are created as follows.
- TREE_CLASSIFIER: Given a dataset it creates a tree of depth - 5 and returns the model it has trained and train datasets predicted value.
- NN_CLASSIFIER: Given dataset trains a neural network model on it and returns the trained model and its predicted output value on training set.
- ADABOOST: It takes dataset, weights of each datapoint of dataset and the classifier type(Ex - tree_classifier or nn_classifier) and returns the model trained, new updated weights and alpha value of the corresponding model.
- RUN_ADABOOST: It takes a list of models, respective alphas and a datapoint and returns the predicted output for the datapoint.
- Using above functions different types of models are created and their output noted.

3.2. Results and Observations

The following types of models are built in the colab and their graph plotted.

- 4 model with first three tree classifiers and last one neural net classifier - It is observed that the train loss generally reduces smoothly for tree classifier but with inclusion of neural net classifier it decreases sharply, overally the train loss decreases with increase in number of learners.
- 15 models all tree - Here number of learners are taken to be 15 and all learners are tree.Train loss and test loss decreases with increase in learners and train accuracy and test accuracy increases with increase in learners. All trees have 5 depth.
- 10 models all neural net - Here number of learners are taken to be 10 and all of them are neural net, train loss and test loss decreases with increase in learners and train accuracy and test accuracy increases with number of learners.
- 5 models all neural net - here number of learners are 5 and all neural net, train loss and test loss decrease with learners, train accuracy and test accuracy increase with learners.

Plot for 10 model neural net case is shown, for other models do refer colab notebooks for graph.

3.3. Problems Faced

If weight values are not updated then the alpha value may sometime become negative and this can give rise to various problems.

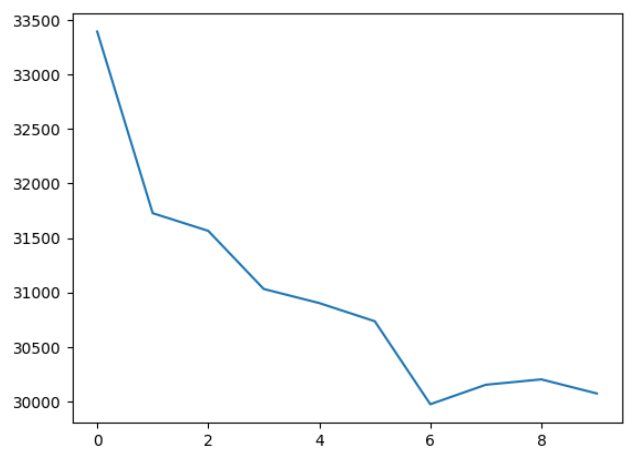


Figure 5. train error vs learners

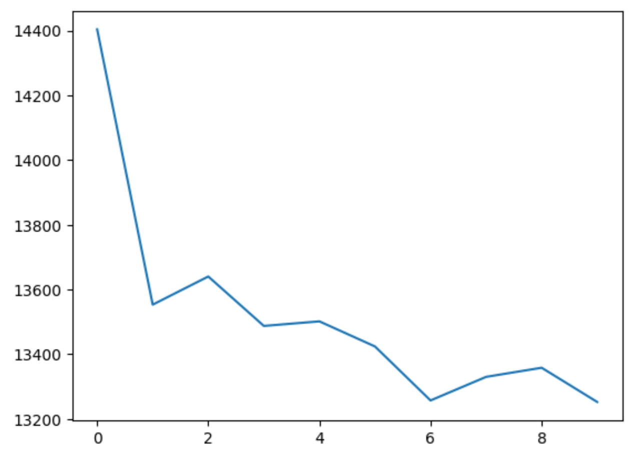


Figure 6. test error vs learners

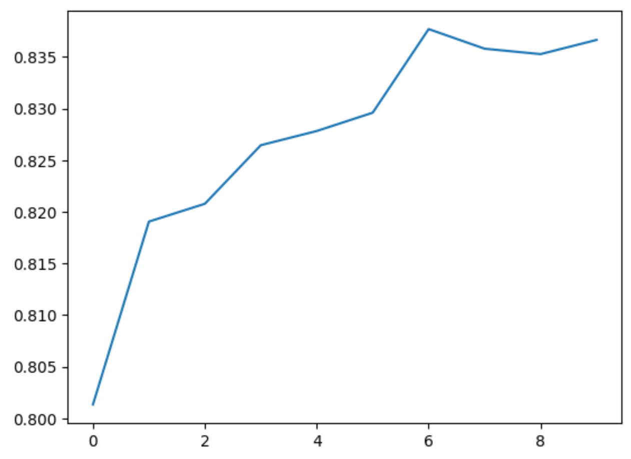


Figure 7. train accuracy vs learners

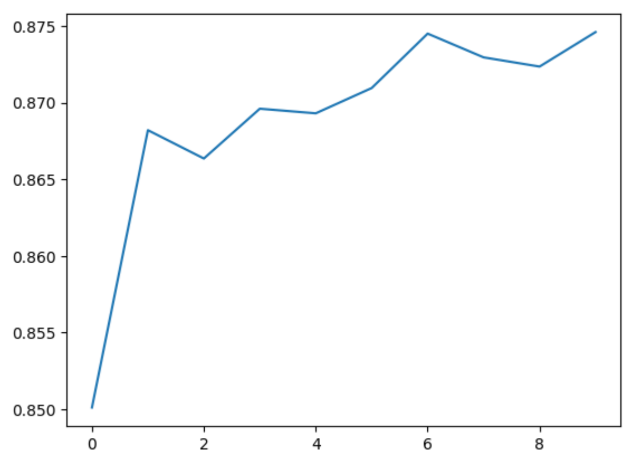


Figure 8. test accuracy vs learners

4. Question 4

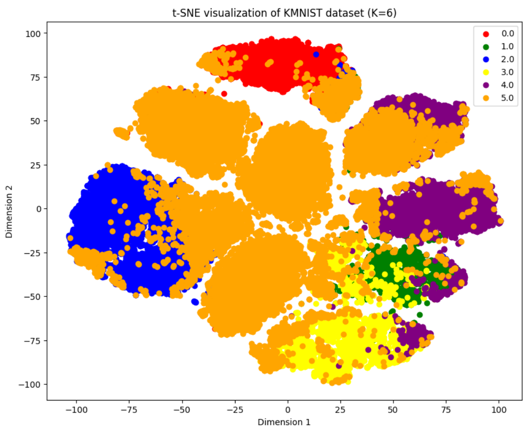
In this question, we implemented two clustering algorithms, GMM (Gaussian mixture models) and KMeans clustering. We tried with different numbers of clusters and calculated mutual normalized information, which is a metric for the quality of clustering.

4.1. Implementation

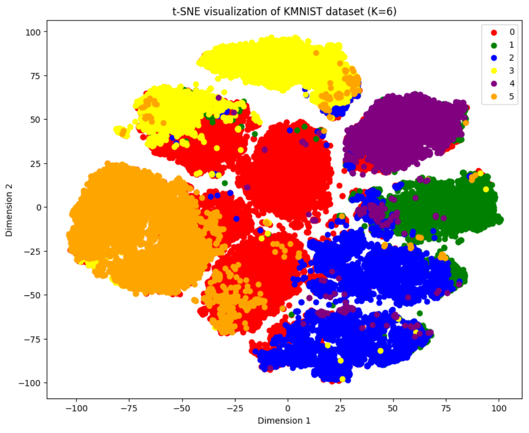
- p4_normalize_Image_dataset: It takes the dataset as input and normalizes it. Normalization leads to good results and better convergence.
- p4_GMM_training: This takes the dataset, number of clusters, and number of epochs as input and returns the mean, variance, and weights of the clusters. With the help of these three, we can calculate the probability of any point belonging to any particular cluster.
- get_labels: This takes the parameters of different Gaussians, datasets, and weights of different Gaussians as input and returns the cluster number for each datapoint (randomly assigned) and the original labels.
- normalized_mutual_info_score: This function takes actual and assigned labels as input and returns normalized mutual information (NMI).
- k_means: This takes the dataset, number of clusters, and number of epochs as input and returns assigned labels (cluster number) for each data point.

4.2. Results and Observations

We tried with 2,6,10,12,14,16,18,20,22,24 number of clusters and observed the clustering using t-SNE plots and using normalized mutual information.

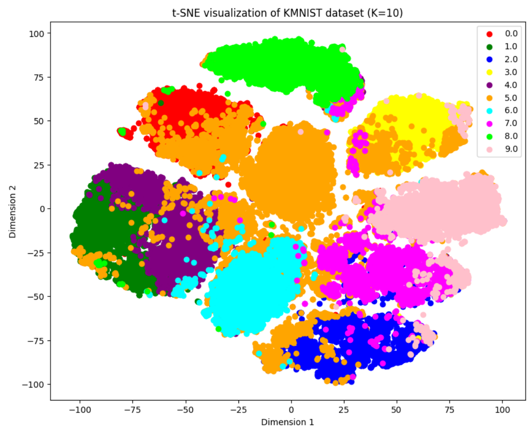


(a) GMM clustering with K = 10

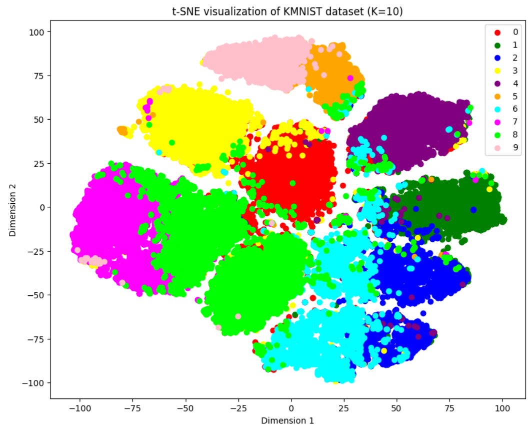


(b) KMeans clustering with K = 6

Figure 9. GMM vs Kmeans for K = 6



(a) GMM clustering with K = 10



(b) KMeans clustering with K = 10

Figure 10. GMM vs Kmeans for K = 10

- We observed that the points which are assigned to a particular cluster by GMM, are almost all assigned to the same cluster by KMeans also. This implies that both algorithms converged to give the same results.

- It is worth noting that in clustering we do not use the original labels as the clustering is used for unlabeled data because of this although the points having the same original labels are expected to be in the same cluster the assigned label can be different from the original labels as we are randomly numbering the clusters, there is no significance of this numbering.

- From the plot of NMI (normalized mutual information) vs the number of clusters, we see almost the same trend in both GMM and KMeans clustering. We see that NMI peaks at K = 10 in both graphs, this is obvious because we are using the KMNIST dataset which has 10 classes thus we expect 10 clusters to be formed and our experiment confirms the same. This means K = 10 is the optimal number of clusters.

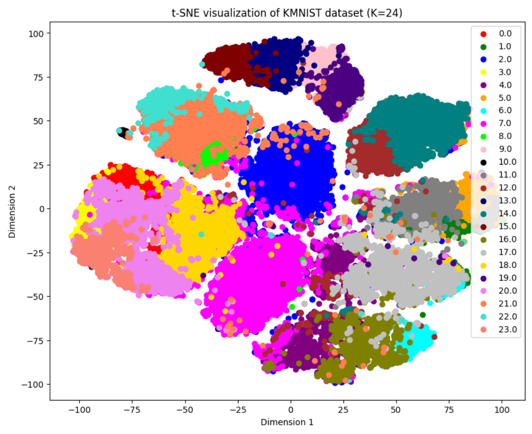
- Although from GMM and KMeans clustering, we get the same results. But from the tSNE plots of GMM and Kmeans, it may seem that two clustering algorithms are not giving the same results, this could be due to,

  1. Because of the random assignment of numbers to the clusters, as we said earlier this does not have any significance.

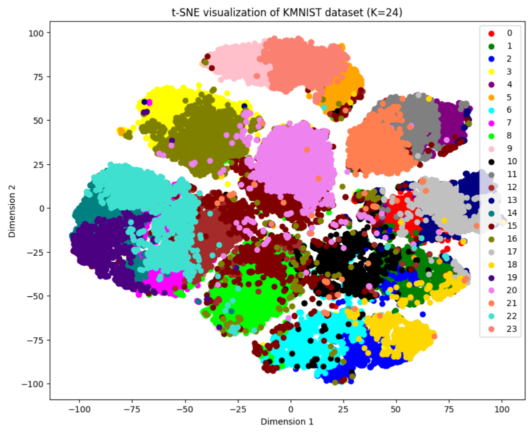
  2. Some clusters overlap in the 2D space and in some plots some cluster is visible above and in some other plots, the same other may not be visible because it hides under some other cluster.

4.3. Problems Faced

With KMNIST data, we face the problem of a singular covariance matrix. This was discussed in detail in the assignment (A-1) where we implemented GMM (Gaussian mixture model).



(a) GMM clustering with K = 24



(b) KMeans clustering with K = 24

Figure 11. GMM vs Kmeans for K = 24

5. Question 5

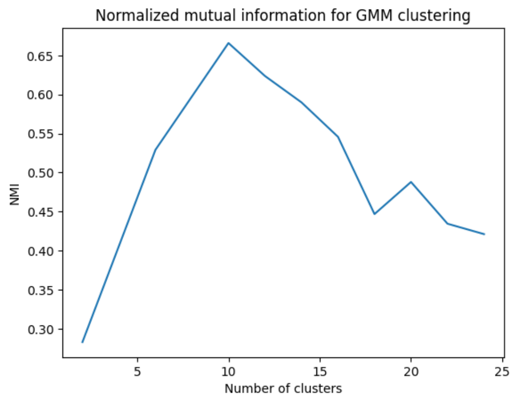
In this question, we have done PCA (Principal Component Analysis) on the KMNIST dataset.

5.1. Implementation

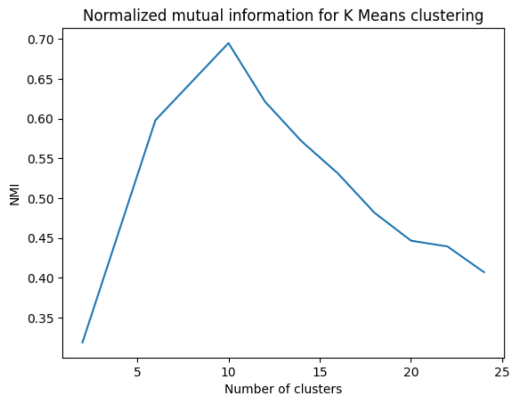
- We first normalize the data.
- Then we calculate the covariance matrix.
- Then we find the eigenvalues and eigenvectors of the covariance matrix.
- Covariance explained can be found as the ratio of the sum of the eigenvalues corresponding to the projected eigenvectors and the sum of the total eigenvalues.

5.2. Results

The trend of the percentage of the covariance explained as a function of the dimension of the projection subspace can be seen in Figures 13.



(a) NMI for GMM clustering



(b) NMI KMeans clustering

Figure 12. NMI vs number of clusters for GMM and KMeans clustering

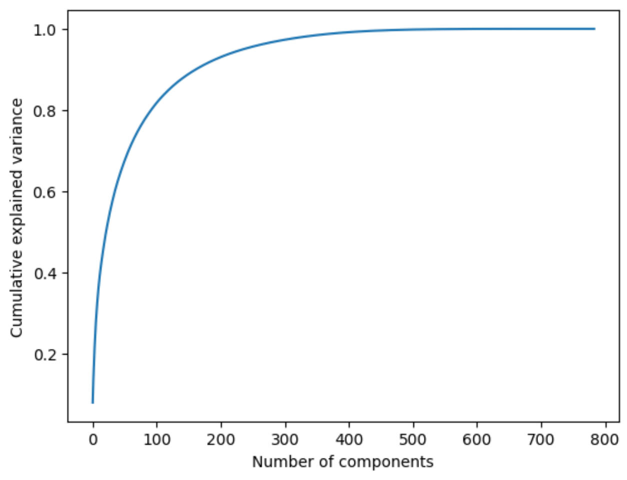


Figure 13. Percentage of covariance explained vs number of dimensions

5.3. Observations

We observed that in order to keep the 90 % covariance preserved we need to project the KMNIT data on the 161-dimensional subspace.