Introduction

Every second, the world generates an unprecedented volume of data. As data has become a crucial component of businesses and organizations across all industries, it is essential to process, analyze, and visualize it appropriately to extract meaningful insights from large datasets. The more data we produce every second, the more challenging it is to analyze and visualize it to draw valid inferences [1]. Many machine learning problems have large number of features, which make training slow, or it requires high end processors to process the data. Slow training may also lead to hardening the process of finding optimal solution, which is refer to as ***'Curse of dimensionality'***. This phenomenon arises when working on analyzing or visualizing data in high dimensional space which do not exist in low dimensional spaces. Reducing the dimensionality of the provided data helps for better understanding of data visualization and speeds up the training. Dimensionality reduction can be approached in 2 ways (1) Projection (2) Manifold learning. Many dimensionalities reduction technique work on manifold modeling where the training instances exist, which is termed as ***Manifold Learning***. It relies on the manifold assumption, also called the manifold hypothesis. When the reduced dataset is provided to the Machine learning classification algorithm, the process of finding the best fit model and parameters would be speeded. Resulting in focusing on improving the model accuracy and F1 score. Classification is a process to separate data into classes and uses the algorithm to identify the category of new observations on the basis of training data. In other words, classification is the process of classifying to which of a set of categories derived from training data set, where categories membership is known, a new observation belongs to.

Background Literature

Dimensionality reduction can be defined as the transformation of data from a high-dimensional space into a low-dimensional space so that the low-dimensional representation retains some meaningful properties of the original data, ideally close to its intrinsic dimension. Working in high-dimensional spaces can be undesirable for many reasons; raw data are often sparse as a consequence of the curse of dimensionality, and analyzing the data is usually computationally intractable [2].
For instance, assume we have a dataset represented in a n × D matrix X consisting of n data vectors X_i ($i \in \{1, 2, \ldots, n\}$) with dimensionality D. Assume further that this dataset has intrinsic dimensionality d (where d < D, and often d << D). Here, in mathematical terms, intrinsic dimensionality means that the points in dataset X are lying on or near a manifold with dimensionality d that is embedded in the D-dimensional space.
The manifold may be non-Riemannian because of discontinuities (i.e., the manifold may consist of several disconnected submanifolds). Dimensionality reduction techniques transform dataset X with dimensionality D into a new dataset Y with dimensionality d, while retaining the geometry of the data as much as possible. In general, neither the geometry of the data manifold, nor the intrinsic dimensionality d of the dataset X are known. Therefore, dimensionality reduction is an ill-posed problem that can only be solved by assuming certain properties of the data (such as its intrinsic dimensionality).

Throughout the paper, we denote a high-dimensional datapoint by X_i , where X_i is the i^{th} row of the D-dimensional data matrix X. The low-dimensional counterpart of X_i is denoted by y_i, where y_i is the i^{th} row of the d-dimensional data matrix Y. Figure 1 shows a taxonomy of techniques for dimensionality reduction. Dimensionality reduction categorized into convex and non-convex techniques. Convex techniques optimize an objective function that does not contain any local optima, whereas non-convex techniques optimize objective functions that do contain local optima. The further subdivisions in the taxonomy are discussed in the review in the following two sections.

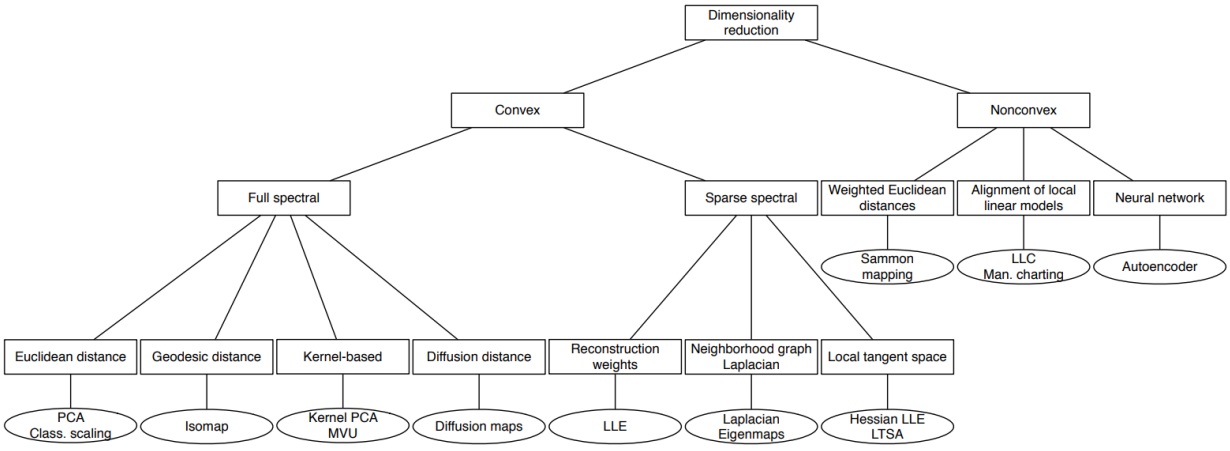


Figure 1: Taxonomy of dimensionality reduction techniques

Project Overview

In this project we focus on reducing the dimension of the datasets provided which consists of classes from 10 to 70 in each dataset. Later, applying the classification algorithm to the reduced extracted features and compared the F-measure and accuracy score of the baseline model and model with reduced features. Brief explanation of the dimensionality reduction techniques utilized, and the classification algorithm used are provided in subsequent paragraphs.

Dimensionality Reduction Techniques

1) Principal Component Analysis (PCA)

Principal Components Analysis (PCA) is a linear technique for dimensionality reduction, which means that it performs dimensionality reduction by embedding the data into a linear subspace of lower dimensionality. Although there exist various techniques to do so, PCA is by far the most popular (unsupervised) linear technique. PCA constructs a low-dimensional representation of the data that describes as much of the variance in the data as possible. This is done by finding a linear basis of reduced dimensionality for the data, in which the amount of variance in the data is maximal [3].

Assumption

Certain assumptions need to be followed to obtain accurate functioning of dimensionality reduction. Below are some of the assumptions needs to be considered:

- Dataset must be linear, i.e., Variables exhibit relationship with other variables in the dataset.
- Pearson correlation coefficient framework is the base to PCA, here initial assumptions was that the axis with high variance is principal components.
- In preprocessing variables are mean normalized to be accessed on same ratio level and PCA assumes that principal component (PC) with high variance is given high priority.
- Outliers (extreme values that deviates from other values) should be less to avoid degradation of ML model.

Data preprocessing

Consider training set data consisting of d-dimensional samples with m- features and perform pre-processing of the provided data to normalize the data in once common scale.
Training set: x (1), x (2), …., x(m)
Preprocessing (feature scaling/ mean normalization):

$$\mu_j = \frac{1}{m}\sum_{i=1}^{m} x_j^{(i)}$$

Replace each $x_j^{(i)}$ with $x_j - \mu_j$

Mean vector $m = \frac{1}{n}\sum_{k=1}^{n} x_k$

If different features on different scales, scale features to have comparable range of values.

Principal Component Analysis (PCA) Algorithm

Calculate covariance matrix by with scaling factor $\frac{1}{N-1}$ to have identical eigenspaces.

$$\Sigma_i = \begin{matrix} \sigma_{11}^2 & \sigma_{12}^2 & \sigma_{13}^2 \\ \sigma_{21}^2 & \sigma_{22}^2 & \sigma_{23}^2 \\ \sigma_{31}^2 & \sigma_{32}^2 & \sigma_{33}^2 \end{matrix}$$

(or) Scatter matrix: $S = \sum_{k=1}^{n}(x_k - m)(x_k - m)^T$

Validate the eigenvector-eigenvalue calculation with the equation, $\Sigma v = \lambda v$
Where, $\sum$-Covariance matrix; v- Eigenvector; λ- Eigenvalue
Sort the eigenvectors by decreasing the eigenvalue, so eigenvector with lowest eigenvalue has less correlation with other variables which can be dropped out. Select high value k (Number of PC) eigenvectors corresponding to eigenvalues. Combine eigenvectors with eigenvalues with largest coefficients to construct $d * k - dimensional$ eigenvector matrix W.

2) Factor Analysis

The objective of factor analysis is to explain variance and covariance within the overall dataset. Factor Analysis creates factors from the observed variables to represent the common variance i.e., variance due to correlation among the observed variables $X$is the

variable and F is the factor and l is the factor loading which is considered as the weight of the factor for the corresponding variable. The number of factors is equal to the number of variables [4].

$$x_i - \mu_i = l_1F_1 + \cdots + l_nF_n + \varepsilon_i$$

The goal is to find out latent factors or hidden groups that are responsible for the observed data in the variables. Identifies similarity between various features and form groups of them which it does by extracting the maximum common variance from all variables.

Steps involved in factor analysis

1) Bartlett's Test of Sphericity:

Bartlett's test determines whether the correlation exists in the data. It proves or disproves the null hypothesis that the correlation matrix is identical. The diagonal members of an identical matrix are all 1. As a result, the null hypothesis asserts that there is no connection between the variables. Factor analysis tries to explain the shared variance; thus, the null hypothesis should be rejected.

2) KMO Test:

KMO test determines how much of the variance between the variables is likely to be common. Greater proportions are expected if there is stronger connection between the variables.

3) Determining the number of factors:

The number of factors can be measured by the amount of shared variance explained by the factors. The amount of variation explained by a factor is measured in eigen values. A handful of factors with eigen values greater than one is chosen.

4) Interpreting the factors:

Loadings, variance, and commonality are used to accomplish this. The loadings of a factor reflect how much it explains a variable. The loading score will be a number between -1 and 1. The fraction of each variable's variance that may be explained by the variables is known as communality.

3) Multidimensional Scaling (MDS)

Multi-Dimension Scaling is a distance-preserving manifold learning method. All manifold learning algorithms assume the dataset lies on a smooth, nonlinear manifold of low dimension and that a mapping $f: R^D -> R^d$ $(D \gg d)$ can be found by preserving one or more properties of the higher dimension space. Distance preserving methods assume that a manifold can be defined by the pairwise distances of its points. In distance preserving methods, a low dimensional embedding is obtained from the higher dimension in such a way that pairwise distances between the points remain same. Some distance preserving methods preserve spatial distances (MDS) while some preserve graph distances [5].

The below table Table 1 shows a comparative study between principal component analysis and multidimensional scaling dimensionality reduction technique.

Features	PCA	PCoA
Method	Canonical vector analysis	Classical scaling
	Geometrical distance between any 2 entities reflects the distance between them and grouping entities reveal the set of generally similar entities	Original scaling method which uses input of similarity/ dissimilarity matrix to generate graphical representation in a small number of dimensions
Data type	Original or quantitative data is used for analysis	Discrete data is used for analysis
Most valuable information	Concentrated in the I principal component	Explained in 1st two or three principal components
Visualizes	Similarity data	Similarity or dissimilarity data
Matrix	Covariance/ variance matrix among samples	Log square distance matrix among samples
Usage	When large number of features are included in the study	Comparatively less data then PCA
Construction utility	Used to construct two- or three-dimensional scatter plot	Used to construct two- or three-dimensional scatter plot
Use cases	Used in cases when there is no missing data and for linear data where more entities than characters scored	Used even when there is no missing and non-linear data and for non-linear data

Table 1: Comparison of PCA and PCoA dimensionality reduction technique

Classification algorithms

K Nearest Neighbor

K Nearest Neighbor algorithm falls under the Supervised Learning category and is used for classification (most commonly) and regression. It is a versatile algorithm also used for imputing missing values and resampling datasets. It considers K Nearest Neighbors (Data points) to predict the class or continuous value for the new Datapoint. K value is determined by finding the minimum distances between the existing features and new point in the feature space for which a class is to be predicted. Distance between the points is calculated by using Euclidean/ Hamming/ Manhattan/ Minkowski distance [6].

Random Forest Classifier

Random Forest Classifier is a bagging technique also known as bootstrap aggregation ensemble technique used in predictions modeling and which uses and built on n- number of decision trees. From the dataset, it performs row and column sampling with replacement which is fed as input to each of the decision trees. It works on voting technique where the new point is categorized into one of the classes based on the majority of votes [7].

Multilayer Perceptron

Multilayer perceptron is a class of feedforward artificial neural network (ANN). It is also known as "vanilla" neural networks. It consists of three types of layers- input layer, output layer and hidden layer. Input layer- introduces input values into the network and it does not have activation function. Hidden layer- perform classification of features. In most cases, two layers is sufficient to solve NN problems. Output layer- functions similar to the hidden layer and the output of the model can be viewed in the output layer. When the n-number of features are provided as input layer which is passed to the hidden layer along with weights w'_{dd} [8].

Radial Basis Function

Radial basis function network is an artificial neural network that uses radial basis functions as activation function. The output of the network is a linear combination of radial basis function of the inputs and neuron parameters. Unlike MLP, which needs at least 1 hidden layer to derive a non-linearity separation. RBNN transforms the input signal into another form, which is feed into neural network to get linear separability. RBNN is composed of input, hidden (or feature vector) and output layer. RBNN constitutes only 1 hidden layer.

Implementation

The section introduces the practice for implementing dimensionality reduction techniques and machine learning algorithms on Google cloud platform such as Google Colab or offline installer such as Visual studio.

In context to the documentation, it is assumed that:

- User primarily using Google cloud services
- Collect the dataset and store it in Google cloud
- Possess basic understanding of Machine Learning and data preprocessing

Recommended tools and products

The Table 2 lists recommended tools and products for each phase of the ML workflow as outlined in this document:

Machine learning workflow step	Recommended tools and products
ML environment setup	• Notebooks • Vertex SDK for Python
ML development	• Cloud Storage • Notebooks
Data processing	• BigQuery • Dataflow • TensorFlow Extended
Operationalized training	• Cloud Storage • PyTorch

Table 2: Recommended tools and products for machine learning workflow

Steps for implementation

Figure 2 shows the overall workflow of Machine Learning implementation. Starting from data preparation we need to perform exploration to deployment for end-to-end Machine Learning implementation.

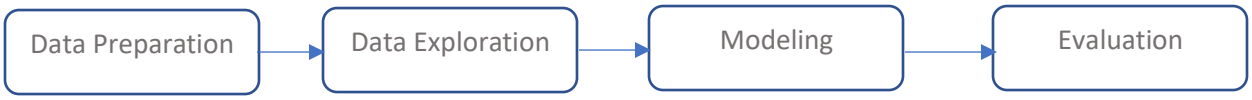


Figure 2: Machine learning model implementation

Data Preparation

Data preparation includes, but is not limited to, data cleansing, labeling the data, dealing with missing data, dealing with inconsistent data, normalization, segmentation, data flattening, data misbalancing, etc. Hand-crafted discriminative features may be also generated during this step if the selected ML approach does not support representation learning from data like in case of end-to-end learning approaches. Feature extraction is the black art of data science. Extracting discriminative features between the different classes or features that remain invariant despite some differences in the raw data of the same class is still unresolved problem in ML. Data collection and preparation are time-consuming processes and constitute of about 80% of the time dedicated for ML-based problem solving.

Data Exploration

Univariate and multivariate analyses should be conducted to generate insights about data separability, linearity, and monotonicity. These insights help in selecting the right ML algorithm given the fact that there is no universally superior ML algorithm according to the no free lunch theorem for machine learning. Statistical machine learning algorithms work under different underline assumptions like the data is separable or the data has a specific distribution or data is balanced or there is no spatial and/or temporal dependency between the data, etc.

Modeling

During the modeling step, ML model is selected, trained, validated and tested. The common approach to build a good model is try to different algorithms and compare their performance. Root-mean-square error (RMSE) is one of the most important criteria used to evaluate the performance of a prediction model. RMSE comprises both the variability of the model (precision) and its bias (accuracy). The two components can be associated with a model's precision (small variance) and its accuracy (small bias). RMSE during training and testing are calculated to figure out the occurrence of under-fitting or over-fitting.

Evaluation

Different quantitative and/or qualitative evaluation metrics should be used to prove the efficacy of the trained model. Model evaluation provides quantitative feedback to optimize the model parameters. The parameters can be optimized manually or using auto-tuning using grid/stochastic search or using evolutionary algorithms or reinforcement learning algorithms as in autoML proposed recently by Google.

Steps to work in Google collaborator

Step 1: Navigate to the Google Collaboratory website

Welcome To Colaboratory - Colaboratory (google.com) visit the website and click on File→ Open new notebook. New notebook will be opened

Step 2: Export dataset

Export the dataset into the notebook, but the dataset will get refreshed when the collaborator is disconnected from the server or if it is unresponsive.

Alternatively, user could save the dataset into google drive and access them as per the requirement, this reduces the running time to upload the dataset on every cycle.
Click on Files → Upload to session storage/ Mount Drive option to upload from local device or cloud storage respectively.

Step 3: Importing libraries

Figure 3 depicts the basic libraries imported to perform the assigned dimensionality reduction techniques and machine learning classification algorithms.

```
[ ]  import numpy as np
     import pandas as pd
     import seaborn as sns
     import matplotlib.pyplot as plt
     %matplotlib inline
     from numpy import array

     from sklearn.preprocessing import StandardScaler
     from sklearn.ensemble import RandomForestClassifier
     from sklearn.metrics import accuracy_score, roc_auc_score, f1_score
     from sklearn.feature_selection import VarianceThreshold
     from sklearn.decomposition import PCA

     from sklearn.model_selection import train_test_split
```

Figure 3: Import in-build libraries

The version of the library packages used in python are provided below

```
Numpy version: 1.20.1
Pandas version: 1.2.4
Matplotlib version: 3.3.4
Scikit Learn version: 0.24.1
Seaborn version: 0.11.1
```

Figure 4: Library version

Experimental Results

Experimental Setting

In general, to perform machine learning modelling, it is essential to import the necessary libraries which are pre-built in the google collaborator such as numpy, pandas, sklearn to name few. Later the format of the file is obtained and read the dataset which serves as input of n- number of features in column and samples as in Figure 5. Features are the independent variable that contribute to the prediction of the dependent variable.

```
[ ] dataset_raw=pd.read_csv('/content/D13.csv',header=None)
    dataset_raw.shape

    (3541, 221)
```

Figure 5: Invoking pandas to read the dataset

As data analysis procedure check out for data which does not contribute to the dependent variable. We checked out for null values and performed one of the three methods to eliminate the null values as per the dataset,

(1) Drop the null values
(2) Fill null values with 0
(3) Fill null values with mean of the column or row

Figure 6 shows the process on how null values are eliminated and checking for other NaN values for verification.

```
# Check for cols with NaN values

for col in dataset_raw.columns:
    if dataset_raw[col].isnull().values.any():
        print(f'NaN in {col}')
```
Python

```
... NaN in 92
```

```
# Checking the column contents descriptive stats

dataset_raw[92].describe()
```
Python

```
... count    0.0
    mean     NaN
    std      NaN
    min      NaN
    25%      NaN
    50%      NaN
    75%      NaN
    max      NaN
    Name: 92, dtype: float64
```

```
# Dropping 92nd feature column

dataset_raw.drop(92, axis=1, inplace=True)
```
Python

```
# Check for other NaN

for col in dataset_raw.columns:
    if dataset_raw[col].isnull().values.any():
        print(f'NaN in {col}')
```
Python

```
print(dataset_raw.shape)                    # before dropping NaN rows
dataset_raw.dropna(inplace=True)            # Will drop rows with NaN val
dataset_raw.shape                           # after dropping NaN rows

# done as a basic check for weeding out NaN values if any
```
Python

Figure 6: Finding NaN values

Later finding NaN values and eliminating it, explored the data by finding the number of classifiers and number of samples provided for each classifier to find the optimal stratified kfold value to avoid

overfitting or underfitting problems in further process. Figure 7 shows the process in which the dataset has been handled.

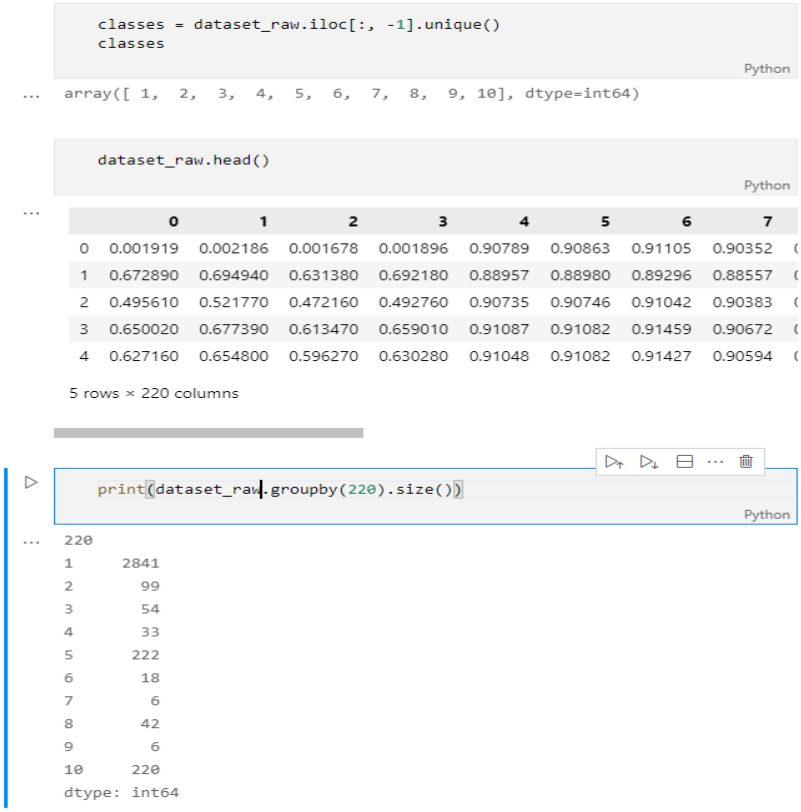


Figure 7: Basic exploration of data

Sub-sampling Process

If the input dataset is found to be very big, 50% of the data is taken as a sample data from each class. This is achieved by the code in the figure 8

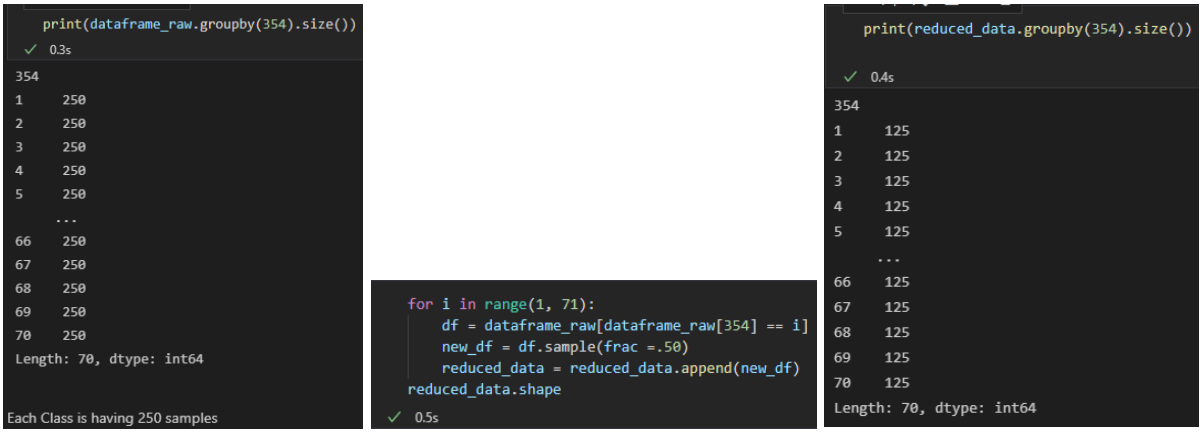


Figure 8: Sub-Sampling Process

Explained Variance Calculation

Explained Variance is the measure of variance of each feature column over the total variance of original dataset. This is calculated using the function defined in the Figure 9.

```
# Function to calculate Explained Variance
def explained_variance(dr_array,dr_arange,Total_Variance):
    #Agruments
    #Dimention reduction array #No. of Columns #Total Variance of the original dataset
    dr_df = pd.DataFrame(dr_array,columns=np.arange(0, dr_arange))
    dr_df_variance = dr_df.var() #calculates variance of features from DR
    Indi_va = []
    for va in dr_df_variance:
        Indi_va.append(va / Total_Variance) #Proportion of variance
    return(Indi_va)
```

✓ 0.2s

Figure 9: Explained Variance Calculation

Principal Component Analysis

To perform PCA, need to ensure that the data to be provided as input should be linearized and scaled. Hence performed standard scaling to the features. To transform data with PCA, instantiated the PCA object, find the principal components using the fit method and applied the rotation and dimensionality reduction by calling transform () method. Figure 10 to achieve 95 % cut-off threshold variance, opted for n_components= 0.95 and performed the PCA to reduce the dimension of the dataset provided.

Transform Data with PCA

We instantiate a PCA object, find the principal components using the fit method, then apply the rotation and dimensionality reduction by calling transform().

We can also specify how many components we want to keep when creating the PCA object.

```
[ ] from sklearn.decomposition import PCA

    #95% of variance
    from sklearn.decomposition import PCA

    pca = PCA(n_components = 0.95)
    pca.fit(scaled_data)
    reduced_pca = pca.transform(scaled_data)
    reduced_pca.shape
```

```
(3541, 33)
```

Explained varianced ratio

```
pca.explained_variance_ratio_
```

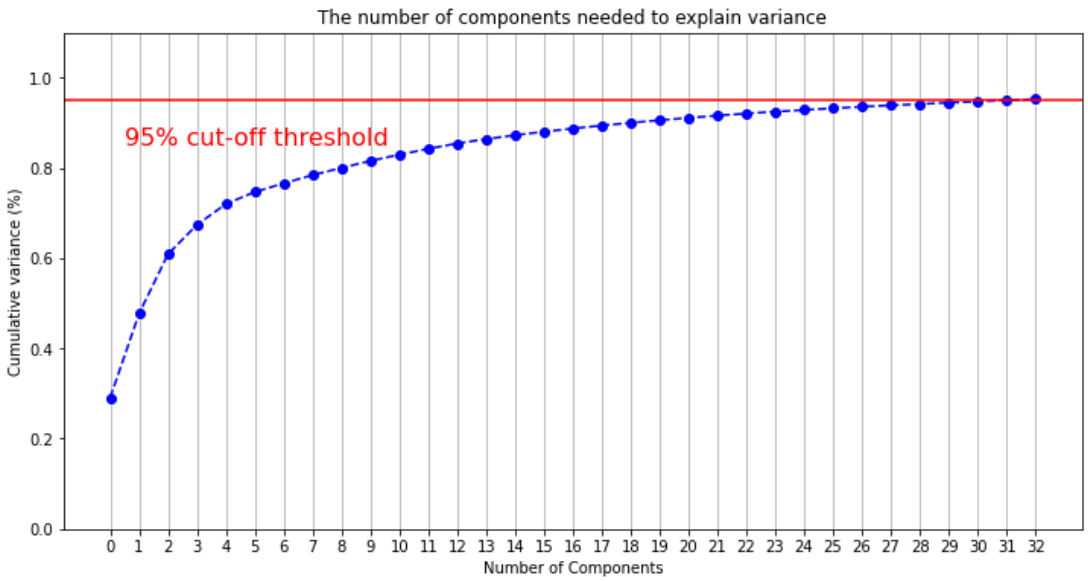
```
array([0.29069028, 0.18637026, 0.13177093, 0.06500434, 0.04622423,
       0.02665718, 0.01911448, 0.01833452, 0.01596356, 0.01539418,
       0.01396421, 0.01286335, 0.01189904, 0.00969206, 0.00858242,
       0.00766134, 0.007158  , 0.00655286, 0.00606104, 0.00571038,
       0.00526438, 0.00509413, 0.00447459, 0.00418755, 0.00378154,
       0.00370346, 0.00344728, 0.00311126, 0.00289133, 0.00285694,
       0.00276209, 0.00265256, 0.00256123])
```

```
[ ] pca_explained_variance_ratio=pca.explained_variance_ratio_.shape
```

```
[ ] pca_explained_variance_ratio
```

```
(33,)
```

Figure 10: Principal Component Analysis



Inference: From the plot we came to conclusion that with 31 to 33 components we could achieve 95% threshold variance for this particular dataset

Plot 1: The number of components needed to explain variance

Factor Analysis

Factor_analyzer Package in python needs to installed before using the below

Bartlett Sphericity Test checks if correlation is present in the given data. This test was done performed for all the datasets before proceeding to Factor Analysis. From the figure 11, we can see that the p value is 0 indicating that we can proceed with factor analysis.

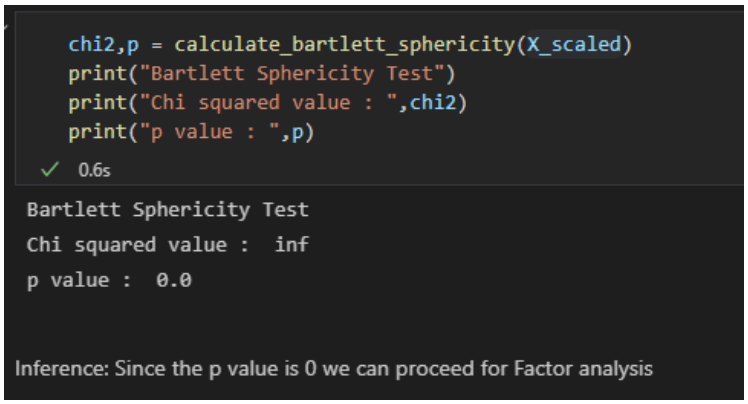


Figure 11: Bartlett Sphericity Test

To Estimate the amount of correlation between the given dataset, KMO Test is performed. From the figure 12, we can see that the value is very close to 1 indicating lot of correlation features which can be reduced.

```
kmo_all,kmo_model = calculate_kmo(X_scaled)
print("KMO Test Statisitc",kmo_model)
```

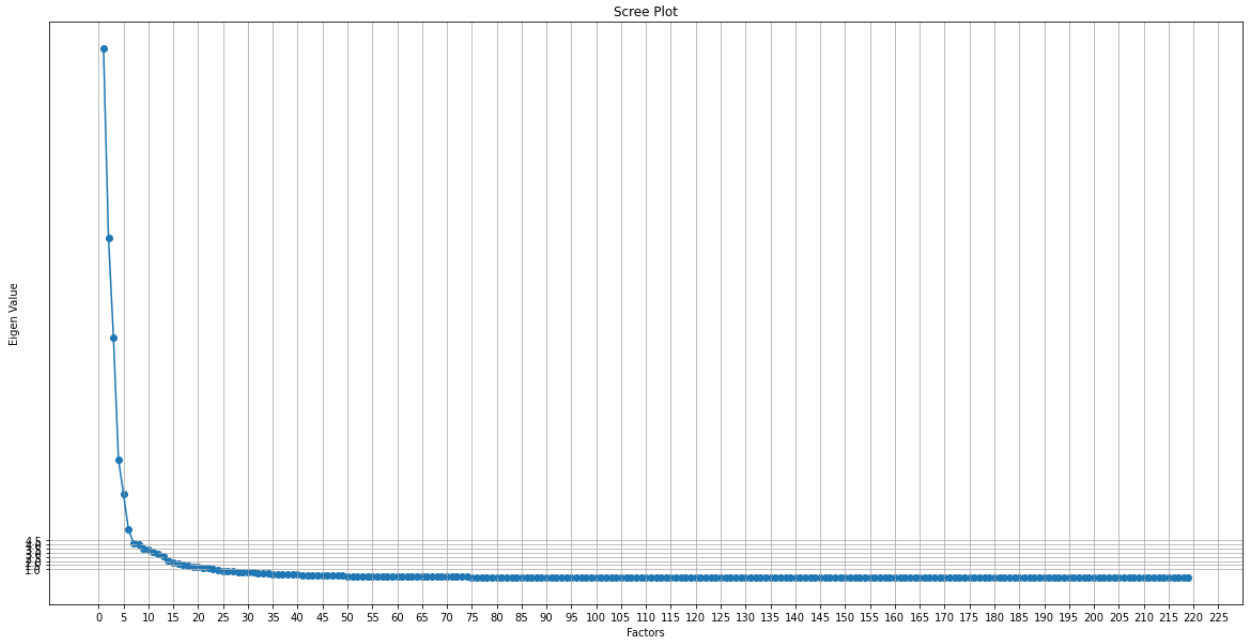
✓ 0.7s

```
KMO Test Statisitc 0.9461936390231579
```

Inference: Since the KMO test value is close to 1, there are many correleated features in the data which can be reduced using Factor Analysis

Figure 12: KMO Test

The Eigen values are calculated and then plotted against the factors. Theoretically, the number of factors more than the eigen value 1 can be considered to extract the features. But, practically, the number of features to be extracted are calculated using trial and error method.



Plot 2: Eigen values against Factors to identify the theoretical factors

Multi Dimensional Scaling

The main purpose of MDS is to preserve the dissimilarities in the reduced dimensionality. Performed multidimensional scaling using default parameters except for the method to find dissimilarity as ‘Euclidean’ distance among the samples Figure 13.

```
from sklearn.manifold import MDS
mds = MDS(dissimilarity='euclidean',random_state=0)
X_transform = mds.fit_transform(scaled_data)
print(X_transform)
```

```
[[13.64126394 -8.16921483]
 [16.91249436 15.12196369]
 [18.01654926  8.17937202]
 ...
 [-1.4746697  -1.45079894]
 [ 2.51161555 -6.97900284]
 [-1.08460871 -3.87156487]]
```

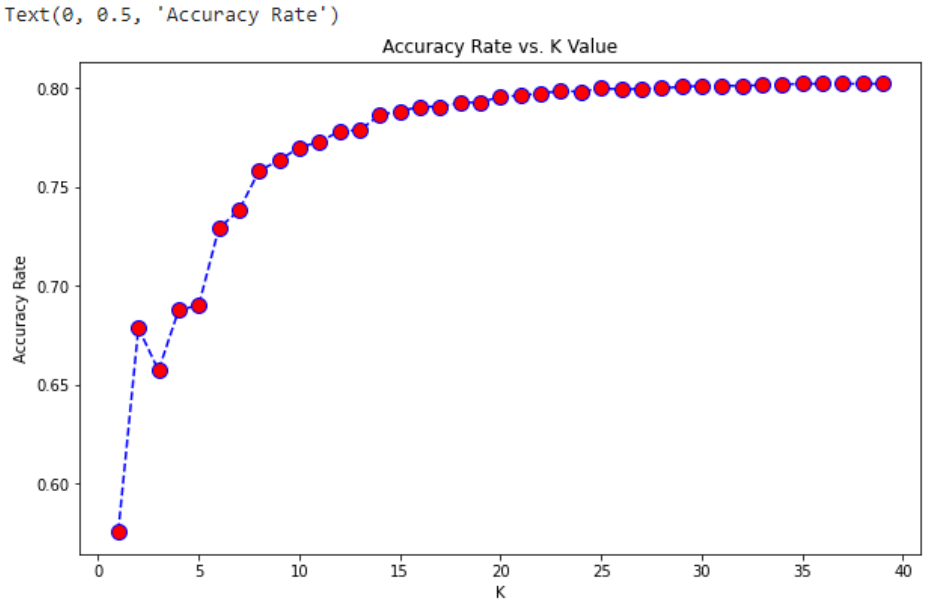
```
[ ] X_transform.shape
```

```
(3541, 2)
```

Figure 13: Multidimensional scaling implementation

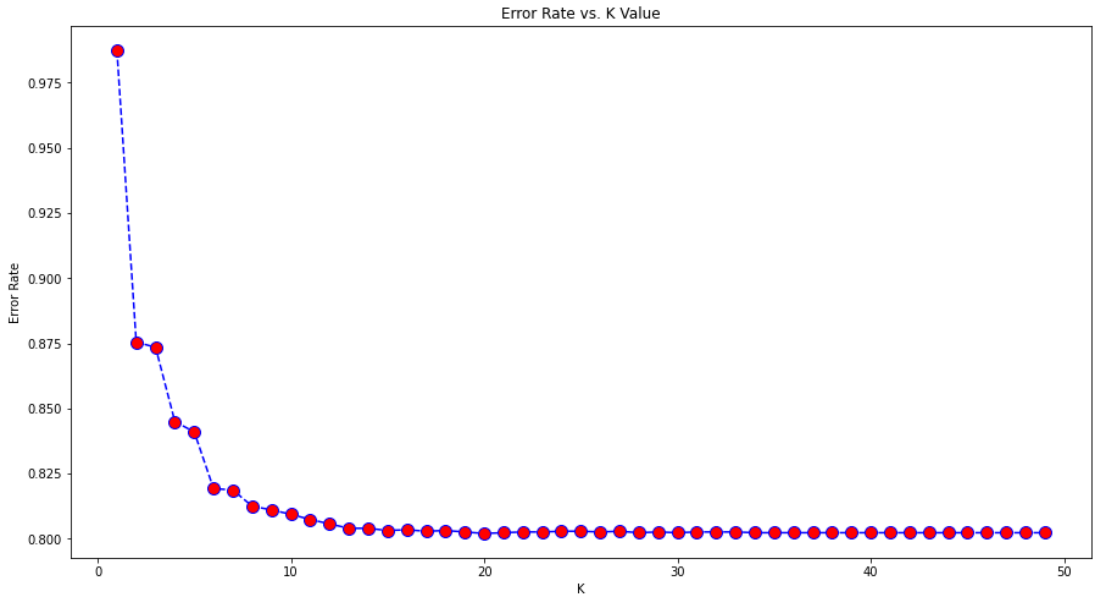
K-Nearest Neighbor

In K-NN, n_neighbors is selected by either of 2 methods (1) Accuracy rate vs K-value (2) Error rate vs K-value. Used for- loop to get accuracy/ error rate of K ranging from 1 to 50 and the rates are stored in as list. Plot 3 and Plot 4 are the resulted accuracy and error rate for the dataset provided.



Plot 3: Accuracy vs K-value plot

```
Text(0, 0.5, 'Error Rate')
```



Plot 4: Error rate vs K-value plot

Identify the lowest error rate and modify the model

```
lowest_error_index = error_rate.index(min(error_rate))+1
print('Minimum Error Rate: ' + str(min(error_rate)))
print('Index position of Minimum Error Rate: ' + str(lowest_error_index))
```

```
Minimum Error Rate: 0.8020333239197966
Index position of Minimum Error Rate: 20
```

Figure 14: Finding lowest error rate

Random Forest Classifier

Performed GridSearchCV to identify the best parameters in the defined range of values Figure 15.

```
#number of trees in RF
n_estimators= [int(x) for x in np.linspace(start=10, stop=100, num=10)]
#Number of features to consider at every split
max_features=['auto','sqrt']
#maximum number of levels in tree
max_depth=[2,4]
#minimum number of samples required to split a node
#min_samples_split=[2,5]
#Minimum number of samples required at each leaf node
#min_sample_leaf=[1,2]
#methods of selecting samples for training each tree
bootstrap=[True, False]
```

```
[ ]  #Create the random grid
     param_grid={'n_estimators': n_estimators,
                 'max_features': max_features,
                 'max_depth': max_depth,

                 'bootstrap': bootstrap}
     param_grid

     {'bootstrap': [True, False],
      'max_depth': [2, 4],
      'max_features': ['auto', 'sqrt'],
      'n_estimators': [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]}

[ ]  RF_Model= RandomForestClassifier()

[ ]  from sklearn.model_selection import GridSearchCV
     RF_Grid= GridSearchCV(estimator=RF_Model, param_grid=param_grid, cv=10, verbose=2, n_j

[ ]  RF_Grid.fit(X_split,y_split)

     Fitting 10 folds for each of 80 candidates, totalling 800 fits
     /usr/local/lib/python3.7/dist-packages/sklearn/model_selection/_split.py:680: UserWarni
       UserWarning,
     GridSearchCV(cv=10, estimator=RandomForestClassifier(), n_jobs=4,
                  param_grid={'bootstrap': [True, False], 'max_depth': [2, 4],
                              'max_features': ['auto', 'sqrt'],
                              'n_estimators': [10, 20, 30, 40, 50, 60, 70, 80, 90,
                                               100]},
                  verbose=2)

[ ]  RF_Grid.best_params_

     {'bootstrap': True, 'max_depth': 4, 'max_features': 'sqrt', 'n_estimators': 10}

[ ]  print(f'Train Accuracy: {RF_Grid.score(X_train,y_train):.3f}')
     print(f'Test Accuracy: {RF_Grid.score(X_test,y_test):.3f}')

     Train Accuracy: 0.802
     Test Accuracy: 0.802
```

Figure 15: (a) (b) GridSearchCV to find best params

Multilayer Perceptron

In Multilayer Perceptron, the mapping between the input and outputs are non-linear. In the below figure, the parameters are tuned to suit the respective dataset. In the figure 16, it can seen that the max_iter is increased to 1000 from the default 200.

```
#Multi-layer Perceptron
from sklearn.neural_network import MLPClassifier
mlp = MLPClassifier(max_iter=1000, hidden_layer_sizes=(50,30,20), activation='relu', random_state=99)
✓ 0.3s
```

Figure 16: Multilayer Perceptron

This is to tackle the below error represented in figure 17. This error rises as the algorithm is optimizing by a stepwise convergent to a minimum and it wasn't found.

multilayer_perceptron : ConvergenceWarning: Stochastic Optimizer: Maximum iterations reached and the optimization hasn't converged yet.Warning?

Figure 17: Convergence error

Apart from this, the number of hidden layers are increased to 3 and neurons in the each layer is also mentioned. The activation function is given as relu which is the rectified linear unit function which returns f(x)

Radial Basis Function

To implement Radial Basis Function Neural Network, the required functions are defined and the modified kmeans function will return the cluster center as well as the standard deviation of the clusters like shown in the below figure. Then the RBFNN is coded as a class. The same beta can be used for the cluster centroids.

```
def kmeans(X, k, max_iters):

    centroids = X[np.random.choice(range(len(X)), k, replace=False)]

    converged = False

    current_iter = 0

    while (not converged) and (current_iter < max_iters):

        cluster_list = [[] for i in range(len(centroids))]

        for x in X:  # Go through each data point
            distances_list = []
            for c in centroids:
                distances_list.append(get_distance(c, x))
            cluster_list[int(np.argmin(distances_list))].append(x)

        cluster_list = list((filter(None, cluster_list)))

        prev_centroids = centroids.copy()

        centroids = []

        for j in range(len(cluster_list)):
            centroids.append(np.mean(cluster_list[j], axis=0))

        pattern = np.abs(np.sum(prev_centroids) - np.sum(centroids))

        converged = (pattern == 0)
        print('K-MEANS: ', int(pattern))
        current_iter += 1

    return np.array(centroids), [np.std(x) for x in cluster_list]
```

Figure 18: kmeans for RBF

```
Fold 1
---------
K-MEANS:  107
K-MEANS:  1
K-MEANS:  1
K-MEANS:  1
K-MEANS:  0
K-MEANS:  1
K-MEANS:  0
K-MEANS:  1
K-MEANS:  0
K-MEANS:  0
K-MEANS:  0
K-MEANS:  0
K-MEANS:  0
K-MEANS:  0
Accuracy:  1.0
f-score:  1.0
```

```
Fold 2
---------
K-MEANS:  426
K-MEANS:  9
K-MEANS:  22
K-MEANS:  1
K-MEANS:  0
K-MEANS:  0
K-MEANS:  1
K-MEANS:  0
K-MEANS:  0
Accuracy:  1.0
f-score:  1.0
```

```
Fold 3
---------
K-MEANS:  172
K-MEANS:  88
K-MEANS:  8
K-MEANS:  0
K-MEANS:  0
K-MEANS:  1
K-MEANS:  0
K-MEANS:  0
K-MEANS:  1
K-MEANS:  0
K-MEANS:  0
K-MEANS:  0
K-MEANS:  0
Accuracy:  1.0
f-score:  1.0
```

```
Fold 4
---------
K-MEANS:  621
K-MEANS:  72
K-MEANS:  0
K-MEANS:  0
K-MEANS:  2
K-MEANS:  0
K-MEANS:  0
K-MEANS:  0
K-MEANS:  0
K-MEANS:  0
Accuracy:  1.0
f-score:  1.0
```

Figure 19: kmeans clustering for each Fold

For this dataset, even the baseline gave perfect Accuracy and F-score. Proving RBFNN is one of the powerful classification method.

Classifier Testing

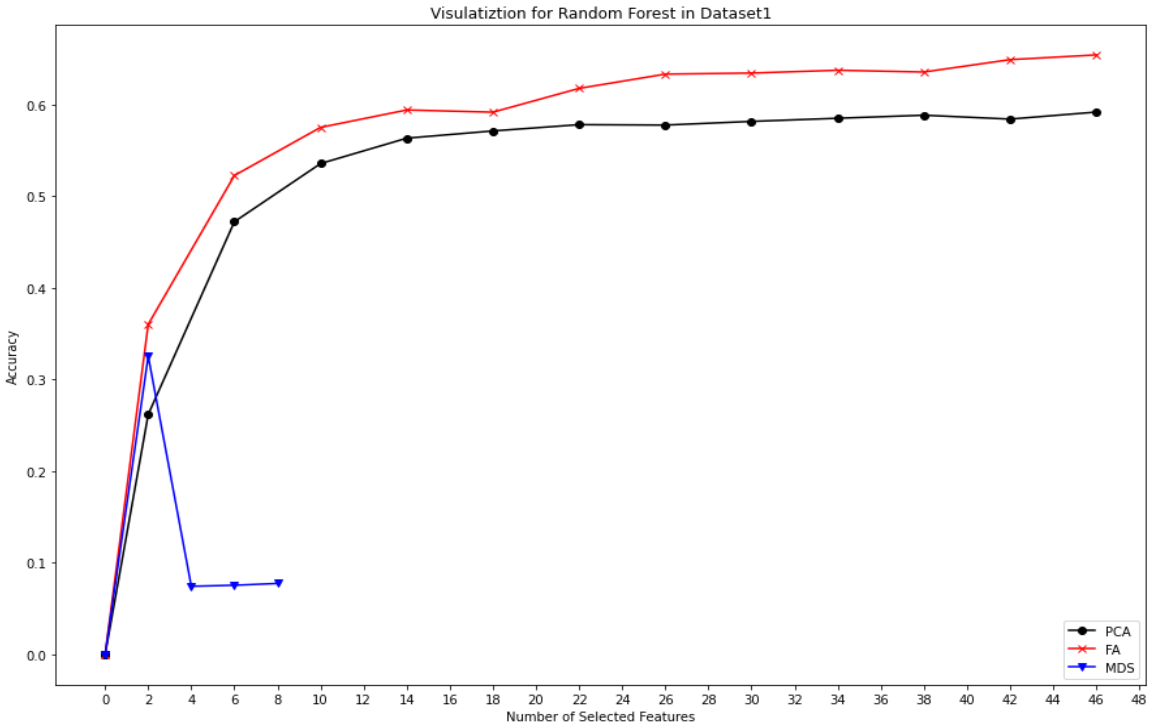
This function is created to check all the DR with the respective Classifier. The number features are increamented by 2 and the accuracy is measured using StratifiedKFold techniques. Once the accuracy a significate amount of accuracy is not increased upon increasing the features, the corresponding number of features is selected for the particular classification model.

```
def classifier_test(model, dr_technique, X, y):

    result = {}
    #X_val = {}

    for i in range(2, 30,2):


        if dr_technique == "FA":
            dr = FactorAnalysis(n_components=i, rotation='varimax', random_state=99)
        elif dr_technique == "PCA":
            dr = PCA(n_components=i, random_state=99, svd_solver='full')
        elif dr_technique == 'MDS':
            dr = MDS(n_components=i, random_state=99)

        X_trans = dr.fit_transform(X)
        scaler = MinMaxScaler()
        X_scaled = scaler.fit_transform(X_trans)

        k_fold = StratifiedKFold(n_splits=10)

        acc = cross_val_score(model, X_scaled, y, cv=k_fold, scoring='accuracy')
        f1 = cross_val_score(model, X_scaled, y, cv=k_fold, scoring='f1_weighted')

        print(f"For {i} features")
        print(f"Accuracy: {acc}")
        print(f"f1-score: {f1}")
        print(f"Mean Accuracy: {acc.mean()}")
        print(f"Mean f1-score: {f1.mean()}")

        result[i] = [acc.mean(), f1.mean()]



        #X_val[i] = X_trans


    return result
```

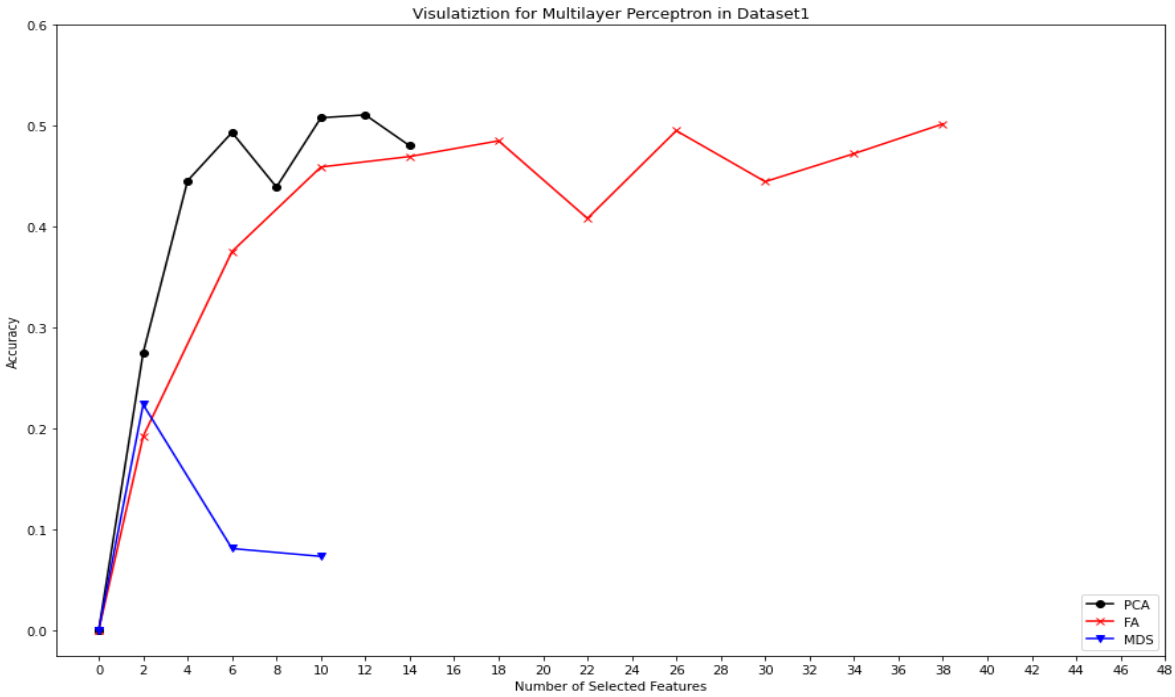
Figure 20: Classifier Testing

Result Analysis

Dimensionality reduction techniques visualization



Plot 5: PCA, FA and MDS techniques with Random Forest Classifier



Plot 6: PCA, FA and MDS techniques with Multilayer Perceptron

- Both plot 5 & 6 are representation for dataset 1. It can be seen from plot 5 that the Factor analysis along with Random Forest has more accuracy than Principle component analysis for the same number of samples.
- For both the classification methods it can see that Multidimentional scaling method performs very poorly after 2 samples. So the number of selected samples are two which in turn is poor. This is also same with other datasets
- 100% Accuracy is achieved for Radial basis function neural network so no other Dimension reduction technique was implemented on the same.
- Sometimes like for Dataset 13, the scores of Random Forest and the scores of KNN seems to be similar.

References

[1] E. Pristine, "Curse of Dimensionality" [Online]. Available: https://www.edupristine.com/blog/curse-dimensionality

[2] "Dimensionality Reduction" [Online]. Available: https://en.wikipedia.org/wiki/Dimensionality_reduction

[3] P. Vadapalli, "Pca in machine learning: Assumptions, steps to apply applications," 2020. [Online]. Available: https://www.upgrad.com/blog/pcain-machine-learning/

[4] D. Babu, "Factor analysis- ml oreo detector," 2020. [Online]. Available: https://towardsdatascience.com/factor-analysis-my-ml-oreodetector-2e02abc2bb30

[5] J. Wang, "Classical multidimensional scaling," Geometric Structure of High-Dimensional Data and Dimensionality Reduction, 2012. [Online]. Available: https://doi.org/10.1007/978-3-642-27497-86

[6] "KNN Algorithm" [Online]. Available: https://www.tutorialspoint.com/machine_learning_with_python/machine_learning_with_python_knn_algorithm_finding_nearest_neighbors.htm

[7] S. E. R., "Understanding random forest," 2021. [Online]. Available: https://www.analyticsvidhya.com/blog/2021/06/understanding-random-forest/

[8] M. Riedmiller., "Multi Layer Perceptrons" [Online]. Available: https://ml.informatik.uni-freiburg.de/former/_media/documents/teaching/ss09/ml/mlps.pdf