

## **BASIC STRUCTURE OF OUR APPROACH(DESIGN)**

1. Train → 60000 with labels
2. Test → 10000 without labels
3. X → only features → 60000
4. Y → only labels → 60000
5. X\_normalized = X/255 → 60000
6. Find Explained variance using X\_normalized → found 42 for 85% variance. [PCA]
7. Fit Transform PCA with n\_components = 42, X\_normalized
8. X\_principal\_components → 60000 pc's
9. Split into train and validate → X\_train, X\_test, y\_train, y\_test
10.
  1. X\_train: y\_train=48000: features and labels
  2. X\_test: y\_test=12000: features and labels
11.
  1. SET 1 → Training Data (48000) → X\_train, y\_train
  2. SET 2 → Validation Data (12000) → X\_test, y\_test
  3. SET 3 → Real Training Data (60000) → X\_principal\_components
12. unlabelled\_test → only features → 10000
13. unlabelled\_test\_normalized → 10000
14. unlabelled\_test\_principal\_components → Applied PCA using same S.D of training (60000) data.
15. APPLY MODELS.
  1. CROSS VAL SCORE
  2. VALIDATION ON 12000. i.e. 20% of Training data.
  3. Confusion Matrix on validation data.
  4. Classification Report on validation data.
  5. ROC curve on validation data.
  6. Classification of real test data i.e. 10000 values.

# **LINEAR AND NON-LINEAR CLASSIFIERS**

- 1. Linear classifiers:** The linear classifiers are the one's which classify the inputs based on linear combination of the feature values or if the classification threshold is linear i.e. a line etc.
  - a. Linear Discriminant Classifier
  - b. Naive Bayes
  - c. Logistic Regression
  - d. Perceptron
  - e. SVM (with linear kernel)
  
- 2. Non-Linear Classifiers:** The non-linear classifiers are the ones which classify the points based on non-linear feature combinations i.e. forming a parabola etc.
  - a. K-Nearest Neighbor
  - b. Random Forest
  - c. Decision Trees
  - d. Multi-layer Perceptron
  - e. Quadratic Discriminant Classifier
  - f. SVM (rbf kernel)

We need to choose the classifiers which are best for our dataset based on whether our dataset is linearly separable or not.

- Dataset having high variance needs feature extraction and the application of non-linear classification method.
- Dataset having low variance can be classified using linear method.

## **Citations:**

- 1.** <https://nlp.stanford.edu/IR-book/html/htmledition/linear-versus-nonlinear-classifiers-1.html>
- 2.** <https://towardsdatascience.com/linear-classifiers-an-overview-e121135bd3bb>

## **1.1: Explanation of Design and Implementation Choices of Model .**

### **1. Underlying Algorithms**

#### **A. Feature Extraction Methods Performed**

1. PCA
2. LDA

#### **B. Classification Methods Performed**

1. SVM
2. Gradient Boosting
3. Random Forest
4. XG Boost
5. KNN
6. Logistic Regression
7. Naive Bayes
8. Decision Tree

#### **ASSUMPTION:**

PCA (n\_components=components explaining 85 to 95% variance) for feature extraction and SVM for classification may work better.

Reason being, Non-linear classification can be performed efficiently using SVM and using appropriate kernel can give great results. SVM is also suitable for large datasets and it performs well when the sparsity is high (0 valued features).

## **2. Description and Working of underlying algorithms(WHY TO CHOOSE)**

### **1. SVM Classifier for Fashion MNSIT**

- A.** SVM constructs the iterative hyperplane in multidimensional space minimizing the error for the separation of different classes. SVM finds the hyperplane which maximizes the margins for clear division of the dataset into labels.

#### **B. Algorithm and Working:**

1. Construct a hyperplane that best divide the available classes in the dataset.
2. Hyperplane should be chosen in the way that maximizes the distance between plane and nearest data points i.e. maximize marginal hyperplane.
3. Distance between nearest points of different classes is called margin. So, we in turn choose right margin for the dataset.
4. For above step choose appropriate value of parameter called C. Larger values of C tends to look for small margin hyperplanes. In other words, we are strictly ordering algorithm/model not to tolerate misclassification of data, but in that process, we will not allow our algorithm/model to generalize.
5. On the other hand, smaller value of C will lead to algorithm/model choose large enough margin hyperplane. Now this may lead to some misclassification of data points, but we allow that so that our algorithm can generalize. In other words, value of C is cost of misclassifying.
6. So, if value C is large the SVM will try to adjust more and more and it can lead to overfitting. Small value of C will give better results with more unknown data points to classify because it will handle noisy data well.
7. Most relevant hyperparameters: {C, kernel}
  - a. **C:** (Default: 1) It is the regularization parameter which is responsible for understanding how much misclassification can be afforded by each training example. It is inversely proportional to the margin between the classes in the hyperplane. Large value of c leads to overfitting the model.
  - b. **Kernel:** (Default: 'rbf') The hyperplane decision boundary between the classes is defined by the kernels. Poly kernel generates new features by using the polynomial combination of the extracted features using PCA components (in our case) whereas 'radial basis function' does the same by finding the distance between a data instance and all the other instances.

### C. Reason to choose for Fashion MNSIT:

1. Most important reason for choosing SVM for our **5-label fashion MNSIT** is the built-in layer which helps SVM interpret the data in a better way which is '**kernel**'. Using 'rbf' kernel will help in better classification and less time complexity. But we will have to find the optimal margin by tuning the best value for 'C'.
2. **Non-linear classification** can be performed efficiently using SVM and it maps the inputs into high-dimensional feature spaces.
3. Using **appropriate kernel function**, SVM can deliver great performance.
4. Negligible risk of over-fitting.
5. SVM is suitable **for large dataset**.
6. **Image classification** can be performed well because SVM performs well when the sparsity is high (0 valued features).
7. Mostly applicable in Face detection, text categorization etc.

### Citations:

1. Chauhan, Vinod. (2016). Re: When we use Support Vector machine for Classification?. Retrieved from: [https://www.researchgate.net/post/When we use Support Vector machine for Classification/57835a4f5b495286d5239d91/citation/download](https://www.researchgate.net/post/When_we_use_Support_Vector_machine_for_Classification/57835a4f5b495286d5239d91/citation/download).
2. <https://www.datacamp.com/community/tutorials/svm-classification-scikit-learn-python>
3. <https://monkeylearn.com/blog/introduction-to-support-vector-machines-svm/>
4. [https://www.researchgate.net/post/Difference between SVM Linear polynomial and RBF kernel](https://www.researchgate.net/post/Difference_between_SVM_Linear_polynomial_and_RBF_kernel)
5. <https://towardsdatascience.com/support-vector-machine-simply-explained-fee28eba5496>

## 2. KNN Classifier for Fashion MNSIT

- A. KNN classifies by estimating the distances between an input  $I$  and all the data points in the dataset, selecting the  $k$  nearest neighbors to  $I$ , finding the mode for labels for these neighbors in case of classification and mean in case of regression.

### B. Algorithm and Working:

1. Load Data
2. Apply standard scaler or divide it by 255(because of pixel values).
3. Extract the features using PCA or LDA (Doesn't work well for our data) or any other feature selection technique.
4. Initialize  $K$  to your chosen number of neighbors by testing training data using CROSS VAL SCORE.
5. For each input  $I$  in the data
  - 5.1 Find the distance between  $I$  and the all the inputs from the data.
  - 5.2 Add the distance and the index of  $I$  and corresponding input index to an ordered collection.
6. Sort the ordered collection of distances and indices in ascending order by the distances.
7. Pick the first  $K$  entries from the sorted collection
8. Get the labels of the selected  $K$  entries
9. If classification (our case), mode of the  $k$  labels is returned.
10. Most relevant hyperparameters:  $\{n\_neighbors, p, weights\}$ 
  - a. **N\_neighbors:** (Default: 5) Number of neighbors to use for queries.
  - b. **Weights:** (uniform) In uniform, all the points in the neighborhood are weighted uniformly whereas in 'distance' points are weighted by the inverse of their distances.
  - c. **P:** (Default: 2) It is the power parameter for the metric.  $P=1$  refers to the Manhattan distance,  $p=2$  refers to the Euclidean distance,

### C. Reason to choose for fashion MNSIT:

1. Decision boundary of KNN is flexible, non-linear and reflects the classes very well. Thus, for Fashion MNSIT with 5 labels, it can end up giving a descent accuracy.
2. Being non-parametric, the data distribution assumptions are not made.
3. Search space robustness. So, with appropriate n\_neighbors and distance metric good classification can be done by KNN for our 5-labelled dataset.
4. For our Fashion **MNSIT dataset**, KNN can be used by tuning it with different hyperparameters for weights ('uniform', 'distance') and p ('Euclidean', 'Manhattan') after the feature extraction and it will classify the data points with decent time complexity. Although, the accuracy may be less than that of SVM which is one of the robust non- linear classifiers when used with kernel 'rbf'.
5. As the classes don't need to be linearly separable for KNN, thus it may result in a descent accuracy.
6. As the performance of the various methods depends on the dataset on which they are being used as well, there has been no method that works better on a given problem. Thus, KNN's accuracy can be considered as the baseline for different algorithms that we will be going to perform.

#### Citations:

1. <https://towardsdatascience.com/machine-learning-basics-with-the-k-nearest-neighbors-algorithm-6a6e71d01761>
2. [https://en.wikibooks.org/wiki/Data\\_Mining\\_Algorithms\\_In\\_R/Classification/kNN](https://en.wikibooks.org/wiki/Data_Mining_Algorithms_In_R/Classification/kNN)
3. <https://towardsdatascience.com/machine-learning-basics-with-the-k-nearest-neighbors-algorithm-6a6e71d017>
4. <https://medium.com/datadriveninvestor/k-nearest-neighbors-in-python-hyperparameters-tuning-716734bc557f>
5. <https://cs231n.github.io/classification/>
6. <http://www.wseas.us/e-library/conferences/2012/CambridgeUSA/MATHCC/MATHCC-18.pdf>

### 3. Decision Tree for Fashion MNSIT- Baseline (Time complexity)

- A.** An easy to interpret white box machine learning algorithm which can be interpreted by checking flow chart formed while making the decisions by the algorithm (Python provides the functionality for the same). Decision tree algorithm works by finding the best attribute for splitting the dataset and continue to do the recursive partitioning until no more attributes or instances are remaining for the subtrees as well.

#### **B. Algorithm and Working:**

1. Attribute selection measure is used to find the best attribute “say B” for splitting the dataset in a logical manner. (A rank is assigned to each feature) ASM can be {‘Information gain’, ‘Gain Ratio’, ‘Gini index’}.
2. Attribute “B” is used as the decision node which breaks the dataset into smaller subsets.
3. Decision tree is formed by following the same procedure recursively until:
  - a. No uncovered attributes are left.
  - b. No more instances left to be incorporated.
  - c. All tuples belong to the same attribute value.
4. Note: Each internal node has a decision rule/attribute that splits the dataset into further subsets.
5. Most relevant hyperparameters: {max\_depth, criterion}
  - a. Max\_depth: Default-None i.e. nodes are expanded until all the leaf nodes contain less than min\_samples\_split samples.
  - b. Criterion: Default- Gini. Gini is used to choose ‘Gini Index’ as the ASM and entropy is used to select ‘Information gain’.

#### **C. Reason to choose for Fashion MNSIT:**

1. With no assumption about the data distribution, decision trees can generalize well. To check whether it works well for Fashion MNSIT dataset, we can use it by tuning it for multiple values of max\_depth and criterion.
2. Decision trees for fashion MNSIT can provide less time complexity because it doesn’t require too much parameters.



3. It can act as a baseline for time complexity for other algorithms to be used to classify the fashion MNIST images.
4. Decision trees can handle multi class problems very well.
5. Although we know that other algorithms can achieve better results in terms of accuracy for image classification which is proved on various image datasets earlier as well. (Ex: satellite image classification etc.)
6. **Note: Decision tree is not recommended for image classification as each pixel is considered as a feature and DT will have to decide and make splits using the best attribute obtained by attribute selection measure (say 'Entropy') to classify the images. In our case, it will not provide better accuracy because of 784-pixel values but we can evaluate it on principal components. Its performance can be compared with Random forest.**

**Citations:**

1. <http://ceur-ws.org/Vol-2320/short7.pdf>
2. <https://cloudxlab.com/blog/fashion-mnist-using-machine-learning/>
3. <https://scikit-learn.org/stable/modules/tree.html>
4. <https://www.datacamp.com/community/tutorials/decision-tree-classification-python>
5. [https://www.researchgate.net/publication/326316293\\_Satellite\\_Image\\_Classification\\_using\\_Decision\\_Tree\\_SVM\\_and\\_k-Nearest\\_Neighbor](https://www.researchgate.net/publication/326316293_Satellite_Image_Classification_using_Decision_Tree_SVM_and_k-Nearest_Neighbor)
6. Nurwauziyah, Iva & Sulistiyah, Umroh & Gede, I & Putra, I Gede Brawiswa & Firdaus, Muhammad. (2018). Satellite Image Classification using Decision Tree, SVM and k-Nearest Neighbor.
7. <https://stackoverflow.com/questions/59327486/image-classification-using-decision-tree>

## 4. Random Forest for Fashion MNSIT

- A.** Random forest is different from the decision tree when it comes to finding the root node, it finds the root node randomly and then creates the multiple decision trees for multiple sample sets. When it comes to prediction it predicts the output from all the DT's and then use the voting to predict the class label.

### **B. Algorithm and Working:**

1. Select the subset of features from all the features. <say 's' from 'f'>.
2. From 's', using best split point find node 'd1'.
3. Split node 'd' into sub nodes using best split.
4. REPEAT [1,4] until all nodes are covered.
5. REPEAT [1,5] until n number of trees are created.
6. Predict the labels.
7. Use voting for finding the final label.

### **In other words:**

1. Take n samples from the dataset.
2. Now from each sample, construct a decision tree.
3. Each tree is used for predicting the class for which the data record belongs.
4. Voting is performed for all the results that are predicted.
5. The result with the greatest number of votes is selected.

**Note:** Relevant hyperparameters: {n\_estimators, criterion}

- a. n\_estimators** (Default: 100) → Total number of trees in the forest.
- b. Criterion** (Default: 'Gini') → Measurement of the quality of the split. It can also be 'entropy' for the information gain whereas 'Gini' is used for the 'Gini Index'.

### C. Reason to choose for Fashion MNSIT:

1. As our **Fashion MNSIT dataset is large enough** and it will end up making **deeper decision trees by using DT algorithm**, we are using Random Forest algorithm which can be computationally expensive, but it will end up giving better accuracy by trading off with the time complexity.
2. It avoids **overfitting and runs very well on large datasets**. Thus, changing the criterion and n\_estimators can provide better results without overfitting.
3. Although it may give better results than the decision tree, but the model will not hold for **the highly imbalanced classes** and its accuracy will deteriorate drastically in such a situation. **But in our case, Fashion MNSIT with only 5 labels can be classified well using Random forest given the proper hyperparameters.**
4. By using both the decision tree and random forest we can analyze the accuracy difference, overfitting (Random forest should not overfit)
5. It is not too much sensitive to parameters used to tune the model and there's no need for tree pruning.

#### Citations:

1. <https://www.datacamp.com/community/tutorials/random-forests-classifier-python>
2. [https://www.stat.berkeley.edu/~breiman/RandomForests/cc\\_home.htm](https://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm)
3. <https://medium.com/@Synced/how-random-forest-algorithm-works-in-machine-learning-3c0fe15b6674>
4. <https://www.cse.huji.ac.il/~daphna/course/CoursePapers/bosch07a.pdf>
5. <http://wgrass.media.osaka-cu.ac.jp/gisideas10/viewpaper.php?id=342>

## 5. Naïve Bayes for Fashion MNSIT - **LINEAR CLASSIFIER** | It will provide the bad results.

### --REASONS TO CHOOSE EXPLAINED--

A. Naïve Bayes is a probabilistic and linear classifier and it looks at the conditional probability distribution i.e probability to assign a class to an input sample. Naive Bayes calculates the conditional probability of all the classes assuming each feature as independent. It is extremely fast as compared to most of the algorithms.

#### B. Algorithm and Working:

1. **Class Probabilities:** Each class probability i.e. Number of samples belonging to class 0 divided by the total number of samples belonging to all the 5 classes.
2. **Conditional Probabilities:** Frequency of each feature value for a given class value divided by the frequency of samples with that class value.
3. **STEPS:**
  - a. Summarize dataset by class.
  - b. Find Class probability.
  - c. Find Conditional probability.
  - d. Thus, training is done.
  - e. Using Bayes Theorem, predictions can be made for the unseen data.

#### C. Reason to choose for Fashion MNSIT:

1. We know that Naïve Bayes is **a linear classifier and it uses conditional probability** for classification, and **it will not generalize well for our Fashion MNSIT dataset**. As if a condition is completely unseen by the model, then it will not correctly classify the test data for that condition as it works on the conditional probability.
2. It also makes **the strong data distribution assumption** that any two features are independent for a class label.
3. In order to see, how linear and probabilistic classifiers perform on the Fashion MNSIT dataset, we are using Naïve Bayes classifier.

#### Citations:

1. <https://arxiv.org/abs/112.0059>
2. <https://towardsdatascience.com/cat-or-dog-introduction-to-naive-bayes-c507f1a6d1a8>

## 6. Logistic Regression for Fashion MNSIT - **LINEAR CLASSIFIER**

### **--REASONS OF CHOOSING EXPLAINED--**

A. Logistic Regression is a predictive analysis algorithm which is based on the concept of probability. It is similar to the linear regression model but with a more complex cost function. It starts with a hypothesis to confine the cost function between  $[0,1]$ .

#### **B. Algorithm and Working:**

1. Load the data.
2. Select the feature as dividing columns in two types of variables-dependent and non-dependent.
3. Split data into train and test sets.
4. Fit the model on train set and predict.
5. Model evaluation using Confusion Matrix.
6. Receiver Operating Characteristic - it is plot of true positive vs false positive.

#### **C. Reason to choose for Fashion MNSIT:**

1. Decision surface of logistic regression is linear, in order to compare how the linear classifiers generalize on our Fashion MNSIT dataset and compare its performance with the Naïve Bayes, we are trying the logistic regression.
2. It is never suitable for the multi class problem. But lbfgs solver can be used for multi class problem.
3. **NOTE: We are using logistic regression to check its classification accuracy with OneVSRest classifier. By using Logistic regression with OneVSRest classifier, it will be making 5 classifiers for the 5-labels in our Fashion MNSIT dataset. i.e. for each label (CLASS 0) it will train a classifier to label class 0 and so on.**
4. The likelihood function will **stop converging once the full separation is achieved which will not happen in our case as our dataset of images will not be separated completely with logistic regression.**

#### **Citations:**

1. <https://machinelearning-blog.com/2018/04/23/logistic-regression-101/>
2. <https://christophm.github.io/interpretable-ml-book/logistic.html>

## 7. Gradient Boosting for Fashion MNSIT

- A. The gradient boosting is the probability approximately correct learning which takes the weak hypothesis or a weak learner and tweaks it to build a strong hypothesis. Gradient boosting algorithm uses decision trees as the weak learner and the trees are added to the model which has the fixed values and doesn't tend to change.

### B. Algorithm and Working:

1. In regression, we calculate the average of target variable.
2. Calculate the residuals. Now this basically means residual equals (actual value - predicted value).
3. Then it constructs a decision tree. The decision tree is made as such each of the leaves contain a prediction which can also be called predicting the residual. Here if number of residuals are more than the leaves then multiple residuals come inside a single leaf. So, in that scenario we take an average of the values and form a simpler decision tree. One thing to note is that it is prediction of the residual not the exact label.
4. Now the samples of data set are passed through the decision tree where residual in the leaves are the predictors for the real labels.
5. **NOTE: So, while training the dataset if we take small steps then overall variance will be less, hence learning rate comes into the picture. So, when learning rate multiplied by the residual is added to average price (In regression), we get the predicted results.**
6. Now the residuals are updated as according the previous steps.
7. Repeat step 3 to step 6 again and again as in accordance to the hyper parameter tuning. For example, number of parameters matches iterations.
8. When training is done all the trees are used in final prediction (average plus all the residuals multiplied by the learning rate)

### C. Reason to choose for Fashion MNSIT:

1. In order to compare the accuracy differences with the decision and random forest and exploring different hyperparameters, we are trying gradient boosting.
2. To analyze the effect of overfitting which was not there in Random forest even when the estimators are increased.

3. We will use Gradient boosting with cross validation to neutralize the overfitting caused by minimizing the error.
4. It can set an upper bound to the time complexity for the classical methods as it may end up making huge number of trees.

**Citations:**

1. <https://towardsdatascience.com/machine-learning-part-18-boosting-algorithms-gradient-boosting-in-python-ef5ae6965be4>
2. <https://www.quora.com/What-are-the-advantages-disadvantages-of-using-Gradient-Boosting-over-Random-Forests>
3. <https://datascience.stackexchange.com/questions/2504/deep-learning-vs-gradient-boosting-when-to-use-what>
4. <https://pdfs.semanticscholar.org/f2e9/7a3ceffebe6bad42af98a2b2be0f42faa9e3.pdf>
5. <https://medium.com/mlreview/gradient-boosting-from-scratch-1e317ae4587d>
6. <https://www.datacamp.com/community/news/gradient-boosting-for-beginners-5ylrejybn9>

## 8. XG Boost for Fashion MNSIT

- A.** It is the advanced version of the gradient tree boosting also known as Xtreme Gradient boosting which reduces the time to converge and improves the performance. It provides the parallel tree boosting. It optimizes the gradient boosting algorithm through parallel processing, tree pruning, handling missing values, and regularization to avoid overfitting and bias.

### **B. Algorithm and Working:**

1. Load the dataset and XGboost library.
2. We then create train and test splits.
3. We need to change the form of data to a form that XGboost can handle. The format is called DMatrix.
4. Then we define XGBoost model. Now here the parameters of the gradient boosting ensemble are defined. Some of them are:
  - a. Objective: Determines the loss function
  - b. num\_class: the number of classes in the dataset
  - c. max\_depth: maximum depth of decision trees that we are training
  - d. eval\_metric: it is evaluation matrix for validation data
5. Finally, the training and validation is done for the dataset.

### **D. Reason to choose for Fashion MNSIT:**

1. The built-in L1 and L2 regularization will prevent the model from overfitting and can end up classifying the images very well. (We will use alpha or lambda as regularizer).
2. XG boost will make the splits until the max\_depth is reached regardless of the loss being negative. It keeps the combined effect.
3. To explore the pros of XGBoost over Gradient Boosting, we are using it to classify our image dataset.
4. It can also run in parallel on various networks and GPU's. this makes it scalable to large datasets.

### **Citations:**

1. <https://towardsdatascience.com/a-beginners-guide-to-xgboost-87f5d4c30ed7>
2. <http://theprofessionalspoint.blogspot.com/2019/03/advantages-of-xgboost-algorithm-in.html>
3. <https://xgboost.readthedocs.io/en/latest/>
4. <https://towardsdatascience.com/https-medium-com-vishalmorde-xgboost-algorithm-long-she-may-rein-edd9f99be63d>



## 9. PCA and LDA for feature extraction

### A. Algorithm and Working (PCA):

1. **Normalization:** This is generally the first step while using PCA for feature selection. This is because PCA is quite sensitive to the variances of the variables in consideration. If ranges for the variables is different then, the variable with longer range is more likely to dominate. So, we do normalization to ensure that the variables in consideration contributes without bias.
2. **Covariance Matrix Calculation:** We compute covariance matrix to analyze the correlation of the variables in consideration. The covariance matrix is  $n \times n$  symmetric matrix where  $n$  relates to the dimensionality of the data set.
3. **Calculate the eigen values and vectors:** Now, we have to construct eigen values and vectors in order to define principal components. Principal components are essentially new features that we form from linear combination of existing features and they are formed in a way that they themselves are completely un-related to each other.
4. **Choosing principal components:** This is also called dimensionality reduction step. We order the principal components in decreasing order of their eigen values because we want to choose those initial components that account for maximum variance in the data and are significant. We then find cumulative explained variance in order to choose the principal components which account for threshold variance (85% in our case) that we deem necessary to account for most of the variation in the data.

### B. Algorithm and Working (LDA):

1. Goal  $\rightarrow$  {Minimize the variation of each class, Maximize the separation of classes.}
2. We have to calculate the within class and between class scatter matrices.
3. Then calculate the eigen vectors and related eigenvalues for the matrices we just calculated.
4. We then sort and select the top values.
5. We then form matrix containing the corresponding eigenvectors.
6. We then find new features by dot product of the calculated matrix and the dataset we have which are essentially called as LDA components and are equal to Total Number of Classes in the original dataset minus 1.

### C. Note:

1. In order to reduce the number of features from 784-pixels, we are using PCA and LDA for feature extraction. After exploring the best method out of two by performing some classification, we will choose one and proceed with the same for feature extraction.

2. PCA removes the correlated features which doesn't add up anything in the decision making.
3. PCA can reduce the overfitting when the classifiers are applied to classify the class labels.
4. LDA is optimal if and only if the classes are Gaussian and have equal covariance.
5. PCA is meant to be used for the features which are strongly correlated.

**Citations:**

1. <https://builtin.com/data-science/step-step-explanation-principal-component-analysis>
2. <https://towardsdatascience.com/linear-discriminant-analysis-in-python-76b8b17817c2>
3. <https://machinelearningmastery.com/linear-discriminant-analysis-for-machine-learning/>
4. <http://www.svcl.ucsd.edu/courses/ece271B-F09/handouts/Dimensionality2.pdf>

## **1.2: Implementation of design choices.**

# **DATA PREPARATION**

## 1. {Importing, Loading, Normalizing} the dataset

```
In [48]: import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
from sklearn.model_selection import train_test_split
import os
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from matplotlib import pyplot as plt
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from sklearn.svm import SVC
from sklearn.model_selection import cross_val_score
import seaborn as sns
import timeit

for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
test=pd.read_csv("../input/ece-657a-w28-asg3-part-1/testX.csv")
train=pd.read_csv("../input/ece-657a-w28-asg3-part-1/train.csv")
sample=pd.read_csv("../input/ece-657a-w28-asg3-part-1/samplesubmission.csv")

/kaggle/input/ece-657a-w28-asg3-part-1/train.csv
/kaggle/input/ece-657a-w28-asg3-part-1/samplesubmission.csv
/kaggle/input/ece-657a-w28-asg3-part-1/testX.csv
```

```
In [147]: X=train.iloc[:,2:786]
y=train.iloc[:,1]
```

```
In [56]: # scaler=StandardScaler() #can also be done by dividing by 255.
# scaler=scaler.fit(X_train)
# X_train_normalized=scaler.transform(X_train)
# X_test_normalized=scaler.transform(X_test)
X_normalized=X/255;
```

```
In [57]: X_normalized
```

Out[57]:

	1	2	3	4	5	6	7	8	9	10	...	775	776	777	778	779
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.000000	...	0.000000	0.000000	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.000000	...	0.000000	0.000000	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.003922	0.000000	...	0.121569	0.035294	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.000000	...	0.000000	0.000000	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.000000	...	0.000000	0.000000	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
59995	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.000000	...	0.000000	0.000000	0.0	0.0	0.0
59996	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.121569	...	0.000000	0.000000	0.0	0.0	0.0
59997	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.000000	...	0.105882	0.000000	0.0	0.0	0.0
59998	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.000000	...	0.000000	0.000000	0.0	0.0	0.0
59999	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.000000	...	0.000000	0.000000	0.0	0.0	0.0

60000 rows × 784 columns

## 2. Finding number of components explaining 85% of variance (PCA) before applying PCA for those number of components.

### a. Finding explained variance ratio:

FINDIND E.V

```
In [58]: pca=PCA(random_state=42)
pca.fit(X_normalized)
explained_variance_ratio=pca.explained_variance_ratio_
```

```
In [59]: explained_variance_ratio=pd.DataFrame(explained_variance_ratio)
```

```
In [60]: explained_variance_ratio
```

Out[60]:

	0
0	2.905760e-01
1	1.769339e-01
2	6.027798e-02
3	4.958797e-02
4	3.845483e-02
...	...
779	1.077699e-07
780	9.408418e-08
781	1.462261e-08
782	7.898940e-09
783	7.462050e-10

**b. Finding cumulative explained variance and choosing the 42 components because 85% variance was explained by 42 components.**

```
In [62]: cum_exp_variance = np.cumsum(explained_variance_ratio)
```

```
In [63]: cum_exp_variance=pd.DataFrame(cum_exp_variance)
```

```
In [64]: cum_exp_variance_percentage=cum_exp_variance*100
```

```
In [65]: cum_exp_variance_percentage
```

```
Out[65]:
```

	0
0	29.057596
1	46.750988
2	52.778786
3	57.737583
4	61.583066
...	...
779	99.999988
780	99.999998
781	99.999999
782	100.000000

```
In [66]: achieved_85=cum_exp_variance_percentage.loc[cum_exp_variance_percentage[0]>85.0,:]  
# achieved_85=(cum_exp_variance[0].gt(95.9999999) & cum_exp_variance[0].lt(95.99334665)).idxmax()
```

```
In [67]: achieved_85
```

```
Out[67]:
```

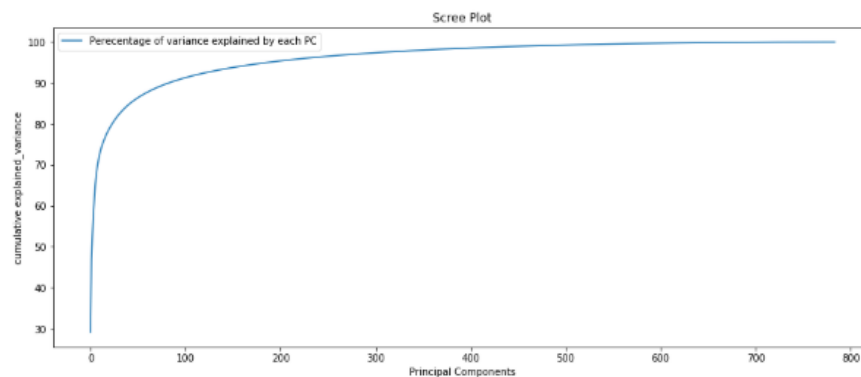
	0
42	85.072217
43	85.255237
44	85.430225
45	85.600768
46	85.769306
...	...
779	99.999988
780	99.999998
781	99.999999
782	100.000000
783	100.000000

742 rows × 1 columns

- c. From the graph, it can also be observed that the 42 principle components i.e. approximately 20% of the total features are explaining the 85% variance of our dataset.

```
plt.figure(figsize=(15,6))
plt.plot(cum_exp_variance_percentage)
plt.title('Scree Plot')
plt.xlabel('Principal Components')
plt.ylabel('cumulative explained_variance')
plt.legend(['Percentage of variance explained by each PC'], loc='best')
```

<matplotlib.legend.Legend at 0x7ff676e4bf68>



### 3. Applying PCA to get the first 42 principal components.....

Finding principal components giving 85% variance i.e 42 components in our case..  
(feature extraction)

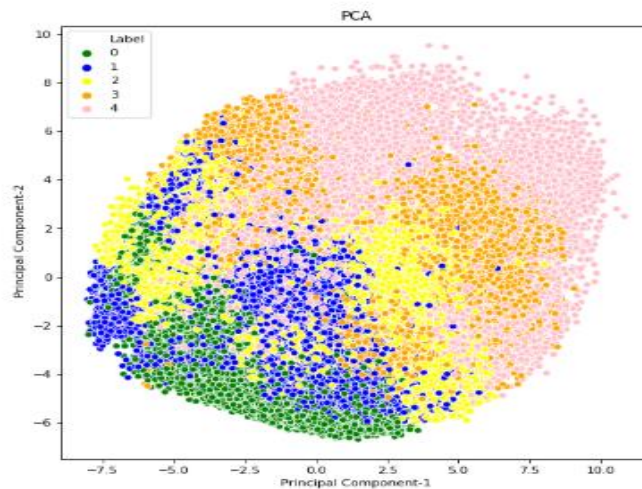
```
pca1=PCA(n_components=comp_giving_85_per_exp_var, random_state=42)
X_principal_components=pca1.fit_transform(X_normalized)
```

```
X_principal_components=pd.DataFrame(X_principal_components)
X_principal_components
```

## 4. Scatter plot pc1 vs pc2

```
In [184]: fig=plt.figure(figsize=(8, 8), dpi= 80)
sns.scatterplot(x=X_principal_components_for_scatter_with_label[0],y=X_principal_components_for_scatter_with_label[1],hue=X_principal_components_for_scatter_with_label.Label,legend='full',palette=['green','blue','yellow','orange','pink'])
plt.xlabel('Principal Component-1')
plt.ylabel('Principal Component-2')
plt.title('PCA')
```

```
Out[184]: Text(0.5, 1.0, 'PCA')
```



## 5. Train test Split:

Splitting the original train data(Given in "train.csv") into train and test in the ratio of 80:20 for making comparisons between different algorithms of: "Feature selection", "Feature Extraction" and "Classification".

```
In [186]: # X=train.iloc[:,2:786]
# y=train.iloc[:,1]
X_train,X_test,y_train,y_test=train_test_split(X_principal_components,y,test_size=0.2,random_state=42)
```

```
In [187]: y_train=pd.DataFrame(y_train)
y_test=pd.DataFrame(y_test)
```

```
In [188]: X_train=X_train.reset_index(drop=True)
X_test=X_test.reset_index(drop=True)
y_train=y_train.reset_index(drop=True)
y_test=y_test.reset_index(drop=True)
```



## 6. Test Data preparation → {Normalizing, Finding principle components}

### Normalizing Test data

```
In [116]:  
unlabelled_test_normalized=unlabelled_test/255
```

```
In [117]:  
unlabelled_test_normalized=pd.DataFrame(unlabelled_test_normalized)
```

```
In [118]:  
unlabelled_test_normalized
```

### FEATURE EXTRACTION PCA

```
In [119]:  
unlabelled_test_principal_components=pcsl.transform(unlabelled_test_normalized)  
unlabelled_test_principal_components=pd.DataFrame(unlabelled_test_principal_components)  
unlabelled_test_principal_components
```

Out[119]:

	0	1	2	3	4	5	6	7	8	9
0	-0.497427	8.408424	-4.735303	0.977229	0.011151	-1.802971	-0.355349	0.793544	-0.145103	-0.09
1	5.523505	-1.779829	-1.031887	1.453044	0.808353	4.989874	-0.803283	-0.893000	-0.590120	-0.83
2	-2.849534	-4.325875	0.411972	0.812824	-0.421293	-0.219940	-0.008051	0.190780	-0.408718	-0.28
3	0.120222	-3.852817	0.794517	1.489391	0.047224	0.713510	-0.435932	0.044258	-1.310890	0.491
4	3.150853	-4.718138	-2.938282	-1.017847	1.801870	-0.594971	0.919010	0.559145	-0.025235	-0.81
...	...	...	...	...	...	...	...	...	...	...
9995	-2.935792	5.143974	-2.988979	-0.110539	0.297547	-0.873208	-0.812878	1.702893	-1.292458	-1.08
9998	-8.951848	-1.528187	2.015984	0.758328	-0.200238	0.084599	-0.180772	1.097710	0.087978	-0.85
9997	4.528835	-2.881831	-3.194938	-0.207922	3.278983	0.103088	-1.181108	0.731790	-0.089000	0.911
9999	-0.953837	-3.711841	1.471084	1.875547	-1.423053	2.848949	-0.554145	-0.425451	-0.784087	0.002
9999	3.842424	2.404902	3.083041	0.953488	-1.379018	-0.428074	0.721589	-0.175299	-0.329427	-0.71

## 7. LDA:.....

## APPLYING LDA AS WELL TO CHECK WHICH ONE WORKS BETTER ON OUR IMAGE DATASET.

**NOTE: WE HAVE PERFORMED ALL THE ALGORITHMS IN DIFFERENT NOTEBOOKS, THUS VARIABLES CAN BE SAME FOR LDA AND PCA IN THIS WORD FILE. SO, WE ARE EXPLICITLY MENTIONING WHICH CLASSIFICATION ALGORITHM IS PERFORMED AFTER WHICH FEATURE EXTRACTION METHOD!!!!!!!**

### a. LDA on Training data

## APPLYING LDA

```
In [11]: from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
lde_4 = LDA(solver='svd') # svd for large no. of features.
lde_transformed_train=lde_4.fit_transform(X.normalized, y) #feature extraction
```

```
In [12]: lde_transformed_train=pd.DataFrame(lde_transformed_train)
```

```
In [13]: lde_transformed_train
```

Out[13]:

	0	1	2	3
0	1.418507	0.845882	2.443755	-2.889882
1	-2.240411	-0.384557	0.979988	0.111109
2	-1.877987	-0.408524	0.553852	0.803048
3	2.784417	2.519599	1.728740	0.873992
4	-1.520178	-0.897940	0.494293	-0.787881
...	...	...	...	...
59995	1.589443	-1.775922	-2.288784	-0.473998
59998	-3.478957	0.747839	-0.222132	0.857417
59997	0.959477	-0.092987	1.400883	-0.045810
59999	-3.827951	1.981441	-0.701571	-0.294759
59999	-1.878245	-0.435905	0.138082	0.348298

60000 rows x 4 columns

## **b. Splitting the dataset into train and validation set.**

### **SPLIT**

```
In [14]: X_train,X_test,y_train,y_test=train_test_split(lda_transformed_train,y,test_size=0.2,
random_state=42)
```

```
In [15]: y_train=pd.DataFrame(y_train)
y_test=pd.DataFrame(y_test)
```

```
In [16]: X_train=X_train.reset_index(drop=True)
X_test=X_test.reset_index(drop=True)
y_train=y_train.reset_index(drop=True)
y_test=y_test.reset_index(drop=True)
```

## **c. Transforming test set:**

### **Normalizing Test data**

```
[19]: unlabelled_test_normalized=unlabelled_test/255
unlabelled_test_normalized=pd.DataFrame(unlabelled_test_normalized)
unlabelled_test_normalized
```

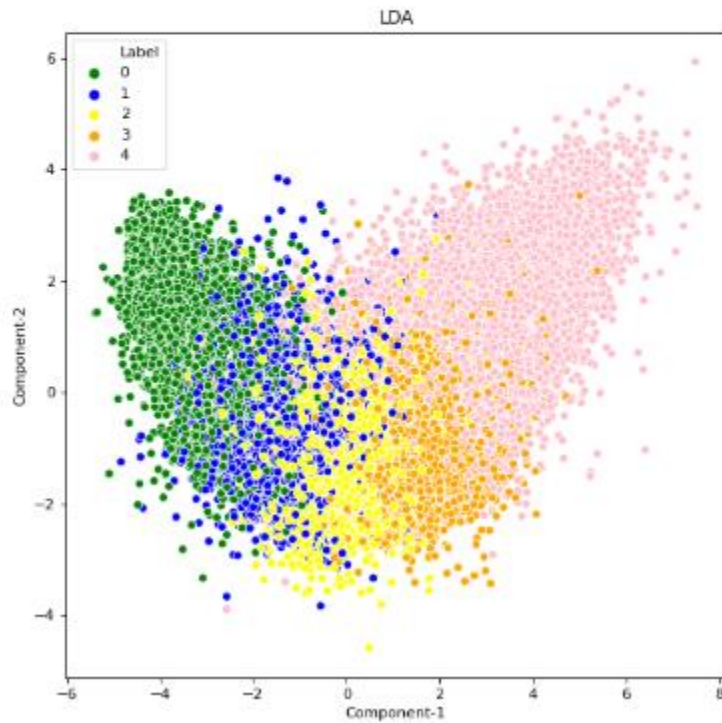
### **FEATURE EXTRACTION LDA**

```
[20]: unlabelled_test_components=lda_4.transform(unlabelled_test_normalized)
unlabelled_test_components=pd.DataFrame(unlabelled_test_components)
unlabelled_test_components
```

## 8. Scatter plot between LD1 vs LD2:

```
In [58]: fig=plt.figure(figsize=(8, 8), dpi= 88)
sns.scatterplot(x=lde_transformed_train[0],y=lde_transformed_train[1],hue=lde_transformed_train.Label,legende='full',palette=['green','blue','yellow','orange','pink'])
plt.xlabel('Component-1')
plt.ylabel('Component-2')
plt.title('LDA')
```

```
Out[58]: Text(0.5, 1.0, 'LDA')
```



## **Note: FOR CODE BLOCK ONLY**

- 1. We have plotted ROC for different parameters as well. To make it simple, we will be sharing the common code block here. Roc curves will be there in last section.**
- 2. We have made confusion matrix and classification report for all the parameters (Sharing for only 1 parameter/ Algorithm) here. Results are in the last section.**
- 3. Same is for all the plots and common steps that we have taken.**
- 4. You can consult the full code on learn.**
- 5. Heat maps and other plots are also plotted which we are not including here in code blocks.**

# **PCA-SVM**

# 1.CROSS VAL SCORE FOR VARYING C.

```
In [ ]:
c_list=[0.1,0.5,1,2,3,5,10,11,12,15,20,30]
mean_accuracy_svm=[]
timetaken1=[]
print("\t\t MEAN ACCURACY AT DIFFERENT C-VALUES\n\n")
for k in c_list:
    start=timeit.default_timer()
    scores=[]
    clf_svm = SVC(kernel='rbf',C=k,degree=3,random_state=42)
    scores_svm=cross_val_score(clf_svm, X_train, np.ravel(y_train), cv=10)
    # dividing the training data of 80% into ratio of 90:10 for multiple splits using one fold for
    testing at a time.
    stop=timeit.default_timer()

    print("Mean Accuracy: ",scores_svm.mean(),"at c= ",k," Time taken ::",stop-start)
    mean_accuracy_svm.append(scores_svm.mean())
    timetaken1.append(stop-start)
```

## 2.Making DataFrame and swarmplot to choose best C with highest accuracy.

Making the dataframe of results

```
df_svm=pd.DataFrame(data=list(zip(c_list,mean_accuracy_svm,timetaken1)),columns=['C v
alues','Mean accuracy SVM','Timetaken SVM'])
```

```
df_svm
```

```
sns.swarmplot( x='C values',y='Mean accuracy SVM', data=df_svm)
plt.title("Accuracy vs C-Values")
```

### 3. RandomSearchCV to select the best C giving C: loguniform (0.1,20) covering continuous values.

Randomsearchcv for svm → resulted 11.7 when used same std dev for everything.  
i.e sd of `x_train`.

```
In [ ]: # from sklearn.model_selection import RandomizedSearchCV
# from sklearn.svm import SVC
# from sklearn.utils.fixes import loguniform
# parameters = {'kernel':('linear', 'rbf', 'sigmoid'), 'C':loguniform(0.1,30),'gamma': loguniform(1e
-4, 1e-3), 'class_weight':('balanced', None)}
# random = RandomizedSearchCV(SVC(), param_distributions=parameters, cv = 5, n_jobs=-1)
# random_result = random.fit(X_train,np.ravel(y_train))
```

```
In [ ]: from sklearn.model_selection import RandomizedSearchCV
from sklearn.utils.fixes import loguniform
parameters = {'kernel':rbf, 'C':loguniform(0.1,20)} # 'gamma': loguniform(1e-4, 1e-3), 'c
lass_weight':('balanced', None) Tried these as well. time complexity was too high.
random = RandomizedSearchCV(SVC(), param_distributions=parameters, cv = 5, n_jobs=-1)
random.fit(X_train,np.ravel(y_train))
print("Best: %f using %s" % (random_result.best_score_, random_result.best_params_))
```

#### 4. Finding the **validation accuracy** by applying the tuned parameters to the validation data.

**a. At C= 10:**

APPLYING THE TUNED PARAMETERS AND PREDICTING THE TEST DATA (20%). 89.64 at c=10

```
clf_svm1 = SVC(kernel='rbf', C=10, degree=3, random_state=42)
clf_svm1.fit(X_train, np.ravel(y_train))

y_pred_svm=clf_svm1.predict(X_test)
accscore_svm = accuracy_score(y_test, y_pred_svm)
```

```
print("Accuracy on 20% Test data after training on best parameters choosen from cross  
valscore i.e c=10::",accscore_svm)
```



## b. At C=11.7:.....

APPLYING THE TUNED PARAMETERS AND PREDICTING THE TEST DATA (20%). 89.69 at c=11.7

```
1: clf_svm11 = SVC(kernel='rbf', C=11.7, degree=3, random_state=42)
   clf_svm11.fit(X_train, np.ravel(y_train))

   y_pred_svm=clf_svm11.predict(X_test)
   accscore_svm = accuracy_score(y_test,y_pred_svm)
```

```
1: print("Accuracy on 20% Test data after training on best parameters choosen from cross
   valscore i.e c=11.7::",accscore_svm)
```

```
Accuracy on 20% Test data after training on best parameters choosen from crossvals
core i.e c=11.7:: 0.8969166666666667
```

## Confusion matrix:.....

Confusion matrix

```
1: from sklearn.metrics import confusion_matrix
   con_mat = confusion_matrix(y_pred_svm, y_test)
   print('Confusion Matrix:\n', con_mat)
```

```
Confusion Matrix:
[[2271 107  28   0   3]
 [ 90 2133 163  44  11]
 [ 18  145 1992 150  31]
 [   0   33  146 2066 120]
 [   0    6   35  107 2301]]
```

## Classification report

```
from sklearn.metrics import classification_report
class_rep = classification_report(y_pred_svm, y_test, target_names = ['Class-0', 'Class-1', 'Class-2', 'Class-3', 'Class-4'])
print('Classification Report:\n', class_rep)
```

```
Classification Report:
              precision    recall  f1-score   support

   Class-0       0.95        0.94        0.95        2489
   Class-1       0.88        0.87        0.88        2441
   Class-2       0.84        0.85        0.85        2336
   Class-3       0.87        0.87        0.87        2365
   Class-4       0.93        0.94        0.94        2449

 accuracy              0.90              12000
 macro avg              0.90              12000
 weighted avg           0.90              12000
```

**c. At C=12:**.....

APPLYING THE TUNED PARAMETERS AND PREDICTING THE TEST DATA (20%). 89.70 at c=12 🔖

```
clf_svm12 = SVC(kernel='rbf', C=12, degree=3, random_state=42, probability=False)
clf_svm12.fit(X_train, np.ravel(y_train))
y_pred_svm=clf_svm12.predict(X_test)
accscore_svm = accuracy_score(y_test,y_pred_svm)
```

```
print("Accuracy on 20% Test data after training on best parameters choosen from cross  
valscore i.e c=12::",accscore_svm)
```

```
Accuracy on 20% Test data after training on best parameters choosen from crossvals  
core i.e c=12:: 0.897
```

## 5. For ROC CURVE→Graphs are in last sec.

ROC AT BEST C=11.7

```
from sklearn.metrics import roc_curve, auc
from sklearn.preprocessing import label_binarize
from sklearn.multiclass import OneVsRestClassifier
from sklearn.metrics import roc_auc_score
y_labelized= label_binarize(y_test, classes=[0,1,2,3,4])
```

```
svm1= OneVsRestClassifier(SVC(C=11.7,probability=False))
svm1.fit(X_train, y_train)
```

```
OneVsRestClassifier(estimator=SVC(C=11.7, break_ties=False, cache_size=200,
                                   class_weight=None, coef0=0.0,
                                   decision_function_shape='ovr', degree=3,
                                   gamma='scale', kernel='rbf', max_iter=-1,
                                   probability=False, random_state=None,
                                   shrinking=True, tol=0.001, verbose=False),
                    n_jobs=None)
```

```
prob = svm1.decision_function(X_test)
```

prob

```
array([[ -0.44278773,  0.47159411, -5.86447256, -3.07403651, -1.72452925],
       [ 1.76573324, -1.19708106, -5.34280897, -4.2979749 , -1.97445032],
       [-7.74708084, -2.2290061 ,  0.52482902, -0.07045938, -5.6785048 ],
       ...,
       [-6.09945821, -6.23588618, -1.65214141, -1.72744427,  1.4729699 ],
       [-2.56943359,  3.38557935, -5.41468336, -3.31904727, -1.56246166],
       [-1.84190016,  2.33683783, -5.23475723, -3.57635238, -3.58687743]])
```

9

```
print('fpr 0', f_p_r[0].shape)
print('fpr 1', f_p_r[1].shape)
print('fpr 2', f_p_r[2].shape)
print('fpr 3', f_p_r[3].shape)
print('fpr 4', f_p_r[4].shape)
print("\n\n")
print('tpr 0', t_p_r[0].shape)
print('tpr 1', t_p_r[1].shape)
print('tpr 2', t_p_r[2].shape)
print('tpr 3', t_p_r[3].shape)
print('tpr 4', t_p_r[4].shape)
```

```
fpr 0 (388,)
fpr 1 (972,)
fpr 2 (1116,)
fpr 3 (1814,)
fpr 4 (488,)
```

```
tpr 0 (388,)
tpr 1 (972,)
tpr 2 (1116,)
tpr 3 (1814,)
tpr 4 (488,)
```

```
print('roc_auc[0]', roc_auc[0])
print('roc_auc[1]', roc_auc[1])
print('roc_auc[2]', roc_auc[2])
print('roc_auc[3]', roc_auc[3])
print('roc_auc[4]', roc_auc[4])
```

```
roc_auc[0] 0.996689761328154
roc_auc[1] 0.9824585528728312
roc_auc[2] 0.9713151348686882
roc_auc[3] 0.9783412815212635
roc_auc[4] 0.9934757348859761
```

```

from itertools import cycle
p=2

plt.figure(figsize=(12,12))
colors = cycle(['aqua', 'darkorange', 'cornflowerblue', 'green', 'black'])
for i, color in zip(range(n_classes), colors):
    plt.plot(f_p_r[i], t_p_r[i], color=color, lw=p, label='ROC curve of class {0} (are
a = {1:0.2f})'.format(i, roc_auc[i]))

plt.plot([0, 1], [0, 1], 'k--', lw=p)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc="lower right")
plt.show()

```

## 6. Overfitting experiment | No leaking of data | Same mean and standard deviation of train data for both the validation set and the real test data.

APPLYING THE TUNED PARAMETERS AND PREDICTING THE TEST DATA (20%). 89.46 at c=30

→ Reason: This value resulted in best accuracy when we applied the normalization and pca after splitting the data into train and val. Giving same s.d as of train data(48000) to the X\_test(val data).

→ PC=79 Explaining 85% variance

That is: NO LEAKING OF INFORMATION EVEN ON VAL DATA.

```

clf_svm30 = SVC(kernel='rbf', C=30, degree=3, random_state=42)
clf_svm30.fit(X_train, np.ravel(y_train))

y_pred_svm_30=clf_svm30.predict(X_test)
accscore_svm_30 = accuracy_score(y_test,y_pred_svm_30)

```

```

print("Accuracy on 20% Test data after training on best parameters choosen from cross
valscore i.e c=30::", accscore_svm_30)

```

## 7.Final data prediction:

TRAINING ON WHOLE DATASET AND TESTING THE UNLABELLED DATA-----

Accuracy for real test data (10000) at c=10 is 89.90

```
#Training
start=timeit.default_timer()
clf_svm2 = SVC(kernel='rbf', C=10, degree=3, random_state=42)
clf_svm2.fit(X_principal_components, np.ravel(y))
```

```
y_pred_svm1=clf_svm2.predict(unlabelled_test_principal_components)
stop=timeit.default_timer()
print('Time_taken', stop-start)
submission_svm_10=y_pred_svm1
```

```
submission_svm_10=pd.DataFrame(submission_svm_10)
submission_svm_10=pd.DataFrame.reset_index(submission_svm_10)
submission_svm_10.columns=['Id', 'Label']
```

```
submission_svm_10
```

```
#Convert DataFrame to a csv file that can be uploaded
#This is saved in the same directory as your notebook
filename = 'submission_svm_10.csv'
submission_svm_10.to_csv(filename, index=False)
print('Saved file:' + filename)
```


Accuracy for real test data (10000) at c=11.7 is 90.06

```
#svm11.7
#Training
start=timeit.default_timer()
clf_svm11 = SVC(kernel='rbf', C=11.7, random_state=42)
clf_svm11.fit(X_principal_components, np.ravel(y))
y_pred_svm11=clf_svm11.predict(unlabelled_test_principal_components)
stop=timeit.default_timer()
print('Time taken ', stop-start)
```

```
Time taken 162.784666243882
```

Accuracy for real test data (10000) at c=12 is 90.06

```
#svm11.7
#Training
start=timeit.default_timer()
clf_svm11 = SVC(kernel='rbf', C=12, random_state=42)
clf_svm11.fit(X_principal_components, np.ravel(y))
y_pred_svm11=clf_svm11.predict(unlabelled_test_principal_components)
stop=timeit.default_timer()
print('Time_taken', stop-start)
```

Using poly kernel at c=10. i.e best c using cross val score. 

Accuracy is 0.87640

```
#svm10 poly
#Training
clf_svm11 = SVC(kernel='poly', C=10, random_state=42)
clf_svm11.fit(X_whole_principal_components, np.ravel(y_full))
y_pred_svm11=clf_svm11.predict(unlabelled_test_principal_components)
```

# **LDA-SVM**



## 1. Cross validation score to choose the best c value and find a comparison with PCA-SVM for the best accuracy.

```
c_list=[0.1,0.5,1,2,3,5,10]

mean_accuracy_svm=[]
timetaken1=[]
print("\t\t\t MEAN ACCURACY AT DIFFERENT C-VALUES\n\n")
for k in c_list:
    start=timeit.default_timer()
    scores=[]
    clf_svm = SVC(kernel='rbf',C=k,degree=3,random_state=42)
    scores_svm=cross_val_score(clf_svm, lda_transformed_train, np.ravel(y), cv=5)
    # dividing the training data of 80% into ratio of 90:10 for multiple splits using one fold for
    testing at a time.
    stop=timeit.default_timer()

    print("Mean Accuracy: ",scores_svm.mean(),"at c = ",k," Time taken ::",stop-start)
    mean_accuracy_svm.append(scores_svm.mean())
    timetaken1.append(stop-start)
```

## 2. Converting into DataFrame and plotting accuracy vs c.

```
df_svm=pd.DataFrame(data=list(zip(c_list,mean_accuracy_svm,timetaken1)),columns=['C values',
'Mean accuracy SVM','Timetaken SVM'])
```

```
df_svm
```

```
sns.swarmplot( x='C values',y='Mean accuracy SVM', data=df_svm)
plt.title("Accuracy vs C-Values")
```

### 3. Applying the best parameters on validation data.

Thus best value of c is 10.

Applying on val data | Best parameters from cross val score |  
accuracy on val data =72.76

```
clf_svm1 = SVC(kernel='rbf', C=10, degree=3, random_state=42)
clf_svm1.fit(X_train, np.ravel(y_train))
y_pred_svm=clf_svm1.predict(X_test)
accscore_svm = accuracy_score(y_test, y_pred_svm)
```

```
print("Accuracy on 20% Test data after training on best parameters chosen from cross  
valscore i.e c=10", accscore_svm)
```

```
Accuracy on 20% Test data after training on best parameters chosen from crossvals  
core i.e c=10: 0.7276666666666667
```

### 4. Confusion matrix

#### Confusion Matrix

```
from sklearn.metrics import confusion_matrix
con_mat = confusion_matrix(y_pred_svm, y_test)
print('Confusion Matrix:\n', con_mat)
```

```
Confusion Matrix:
[[1992  264   32    0    3]
 [ 369 1726  559   43   39]
 [   15  378 1392  415   58]
 [    1   31  318 1598  342]
 [    2   25   63  319 2832]]
```

## 5. Classification Report

### Classification Report

```
from sklearn.metrics import classification_report
class_rep = classification_report(y_pred_svm, y_test, target_names = ['Class-0', 'Class-1', 'Class-2', 'Class-3', 'Class-4'])
print('Classification Report:\n', class_rep)
```

```
Classification Report:
              precision    recall  f1-score   support

   Class-0       0.84      0.87      0.85       2291
   Class-1       0.71      0.63      0.67       2736
   Class-2       0.59      0.62      0.60       2250
   Class-3       0.67      0.70      0.68       2282
   Class-4       0.82      0.83      0.83       2441

 accuracy              0.73       12000
 macro avg              0.73       12000
 weighted avg           0.73       12000
```

## 6. For ROC → Graphs are in analysis section

```
ldsv1 = OneVsRestClassifier(SVC(kernel='rbf', C=10, degree=3, random_state=42))
ldsv1.fit(X_train, y_train)
```

```
OneVsRestClassifier(estimator=SVC(C=10, break_ties=False, cache_size=200,
                                   class_weight=None, coef0=0.0,
                                   decision_function_shape='ovr', degree=3,
                                   gamma='scale', kernel='rbf', max_iter=-1,
                                   probability=False, random_state=42,
                                   shrinking=True, tol=0.001, verbose=False),
                    n_jobs=None)
```

```
prob = ldsv1.decision_function(X_test)
```

## 7. Testing the final dataset

TRAINING ON WHOLE DATASET AND TESTING THE UNLABELLED DATA-

```
# Training
start=timeit.default_timer()

clf_svm2 = SVC(kernel='rbf', C=10, degree=3, random_state=42)
clf_svm2.fit(lda_transformed_train, np.ravel(y))
y_pred_svm1=clf_svm2.predict(unlabelled_test_components)

stop=timeit.default_timer()
```

```
print('Time Taken', stop-start)
```

```
Time Taken 96.43273292500089
```

```
submission_svm_10=y_pred_svm1
```

```
submission_svm_10=pd.DataFrame(submission_svm_10)
submission_svm_10=pd.DataFrame.reset_index(submission_svm_10)
submission_svm_10.columns=['Id', 'Label']
```

# **PCA- GRADIENT BOOSTING**

## 1. CROSS VAL SCORE with different hyperparameters. {Estimators and learning rate only} → depth – 3 (default)

```
In [ ]:
number_of_estimators=[5,10,25,50,100]
learning_rate=[0.001,0.01,0.05,0.1,1]
for i in number_of_estimators:
    for j in learning_rate:
        scores=[]
        gb_clfier_1 = GradientBoostingClassifier(n_estimators=i, learning_rate=j, loss
        ='deviance', random_state=42)
        scores=cross_val_score(gb_clfier_1, X_principal_components, y, cv=5)
        print("Mean Accuracy: ", scores.mean(), "at number of estimators= ", i, "learnin
        g rate=", j)
```

## 2. CROSS VAL SCORE with different hyperparameters. {Estimators and learning rate and max\_depth}

```
lr = [0.01, 0.1, 1]
n_estimators = [10,50,100]
max_depth = [10,20]

for l in lr:
    for e in n_estimators:
        for d in max_depth:

            start=timeit.default_timer()

            gb2 = GradientBoostingClassifier(loss='deviance', n_estimators= e, learni
            ng_rate= l, max_depth = d, random_state=42 )
            scores= cross_val_score(gb2,X_principal_components,y,cv = 5)
            accscore=scores.mean()
            stop=timeit.default_timer()

            print('Mean Accuracy at ', 'learning rate=', l, ' , n_estimators=', e, ' and a
            t max_depth=', d, 'is', accscore, ' where Timetaken is ', stop-start)
```

### 3. Best parameters:

RESULT→

BEST parameters using cross val score are:

→learning rate=0.1

→n\_estimators= 100

→max\_depth=10

### 4. Applying on validation data

Validation data with best parameters

```
start=timeit.default_timer()

gb3 = GradientBoostingClassifier(loss='deviance', n_estimators= 100, learning_rate=
0.1, max_depth = 10, random_state=42 )
gb3.fit(X_train,np.ravel(y_train))
y_pred=gb3.predict(X_test)

stop=timeit.default_timer()
```

```
accscore=accuracy_score(y_pred,y_test)
print('Mean Accuracy at ', 'learning rate=',0.1, ' , n_estimators=',100, ' and at max_de
pth=',10, 'is', accscore, ' where Timetaken is ', stop-start)
```

```
Mean Accuracy at learning rate= 0.1 , n_estimators= 100 and at max_depth= 10 is
0.88275 where Timetaken is 1422.0207228660001
```

## 5. Confusion matrix

### Confusion Matrix

```
from sklearn.metrics import confusion_matrix
con_mat = confusion_matrix(y_pred, y_test)
print('Confusion Matrix:\n', con_mat)
```

```
Confusion Matrix:
[[2249   92   41    8    9]
 [  97 2114  158   47   13]
 [  31  169 1978  183   54]
 [    1   39  154 2000  130]
 [    1   10   49  137 2260]]
```

## 6. Classification Report

### Classification Report

```
from sklearn.metrics import classification_report
class_rep = classification_report(y_pred, y_test, target_names = ['Class-0', 'Class-1', 'Class-2', 'Class-3', 'Class-4'])
print('Classification Report:\n', class_rep)
```

```
Classification Report:
              precision    recall  f1-score   support

   Class-0       0.95      0.94      0.94       2391
   Class-1       0.87      0.87      0.87       2421
   Class-2       0.83      0.82      0.83       2407
   Class-3       0.84      0.86      0.85       2324
   Class-4       0.92      0.92      0.92       2457

 accuracy              0.88              12000
 macro avg           0.88      0.88      0.88              12000
weighted avg           0.88      0.88      0.88              12000
```



## 7. For ROC curve → Graphs are in analysis section

```
gb4= OneVsRestClassifier(GradientBoostingClassifier(loss='deviance', n_estimators= 100, learning_rate= 0.1, max_depth = 10, random_state=42 ))
gb4.fit(X_train, y_train)
```

```
OneVsRestClassifier(estimator=GradientBoostingClassifier(ccp_alpha=0.0,
                                                         criterion='friedman_mse',
                                                         init=None,
                                                         learning_rate=0.1,
                                                         loss='deviance',
                                                         max_depth=10,
                                                         max_features=None,
                                                         max_leaf_nodes=None,
                                                         min_impurity_decrease=0.0,
                                                         min_impurity_split=None,
                                                         min_samples_leaf=1,
                                                         min_samples_split=2,
                                                         min_weight_fraction_leaf=0.0,
                                                         n_estimators=100,
                                                         n_iter_no_change=None,
                                                         presort='deprecated',
                                                         random_state=42,
                                                         subsample=1.0,
                                                         tol=0.0001,
                                                         validation_fraction=0.1,
                                                         verbose=0,
                                                         warm_start=False),
                    n_jobs=None)
```

## 8. Predicting Unlabeled test data

```
start=timeit.default_timer()

gb5 = GradientBoostingClassifier(loss='deviance', n_estimators= 100, learning_rate= 0.1, max_depth = 10, random_state=42 )
gb5.fit(X_principal_components, np.ravel(y))
y_pred=gb5.predict(unlabelled_test_principal_components)

stop=timeit.default_timer()
print('Time Taken', stop-start)
```

```
Time Taken 1934.9617752699996
```

# **LDA- GRADIENT BOOSTING**

# 1. Applying the best parameters obtained from the previous combination for LDA-Gradient Boosting.

## GRADIENT BOOSTING

Directly applying the default parameters with depth=10, lr=[0.1,1] {result obtained from PCA}

```
In [ ]:
from sklearn.ensemble import GradientBoostingClassifier
lr = [0.1,1]
n_estimators = [100]
max_depth = [10]

for l in lr:
    for e in n_estimators:
        for d in max_depth:

            start=timeit.default_timer()

            gb2 = GradientBoostingClassifier(loss='deviance', n_estimators= e, learning_rate=
1, max_depth = d, random_state=42 )
            gb2.fit(X_train,np.ravel(y_train))
            y_pred=gb2.predict(X_test)
            accscore=accuracy_score(y_pred,y_test)
            stop=timeit.default_timer()

            print('Mean Accuracy at ', 'learning rate=',l, ' , n_estimators=',e, ' and at max_dept
h=',d,'is', accscore, ' where Timetaken is ', stop-start)
```

# 2. Reapplying the best parameters on the validation set and Confusion matrix for the best result.

BEST PARAMETERS:: 📌

1.DEPTH=10, LEARNING RATE =0.1 , N\_ESTIMATORS=100

## CONFUSION MATRIX

```
from sklearn.ensemble import GradientBoostingClassifier

start=timeit.default_timer()

gb2 = GradientBoostingClassifier(loss='deviance',n_estimators= 100,learning_rate= 0.1, max_depth = 10, random_state=42 )
gb2.fit(X_train,np.ravel(y_train))
y_pred=gb2.predict(X_test)
accscore=accuracy_score(y_pred,y_test)

stop=timeit.default_timer()

print('Mean Accuracy at ', 'learning rate=',l,' , n_estimators=',e,' and at max_depth=',d,'is',accscore, ' where Timetaken is ',stop-start)# Confusion Matrix

from sklearn.metrics import confusion_matrix
con_mat = confusion_matrix(y_pred, y_test)
print('Confusion Matrix:\n', con_mat)
```

## 3. Confusion Matrix:

```
Mean Accuracy at learning rate= 0.1 , n_estimators= 100 and at max_depth= 10 is 0.7140833
3333333333 where Timetaken is 161.147237940997
Confusion Matrix:
[[2008  319   41    0    4]
 [ 338 1616  549   40   39]
 [  27  419 1342  410   53]
 [   2   42  366 1576  343]
 [   4   28   66  341 2027]]
```

## 4. Classification Report:

## Classification Report

```
In [72]: from sklearn.metrics import classification_report
class_rep = classification_report(y_pred, y_test, target_names = ['Class-0', 'Class-1', 'Class-2', 'Class-3', 'Class-4'])
print('Classification Report:\n', class_rep)
```

Classification Report:				
	precision	recall	f1-score	support
Class-0	0.84	0.85	0.85	2372
Class-1	0.67	0.63	0.65	2582
Class-2	0.57	0.60	0.58	2251
Class-3	0.67	0.68	0.67	2329
Class-4	0.82	0.82	0.82	2466
accuracy			0.71	12000
macro avg	0.71	0.71	0.71	12000
weighted avg	0.71	0.71	0.71	12000

**5. For Roc curve→ Graph is in the analysis section.**

```
In [77]: nb1= OneVsRestClassifier(GradientBoostingClassifier(loss='deviance',n_estimators= 100,learning_
rate= 0.1, max_depth = 10, random_state=42 ))
nb1.fit(X_train, y_train)
```

```
Out[77]: OneVsRestClassifier(estimator=GradientBoostingClassifier(ccp_alpha=0.0,
                                                             criterion='friedman_mse',
                                                             init=None,
                                                             learning_rate=0.1,
                                                             loss='deviance',
                                                             max_depth=10,
                                                             max_features=None,
                                                             max_leaf_nodes=None,
                                                             min_impurity_decrease=0.0,
                                                             min_impurity_split=None,
                                                             min_samples_leaf=1,
                                                             min_samples_split=2,
                                                             min_weight_fraction_leaf=0.0,
                                                             n_estimators=100,
                                                             n_iter_no_change=None,
                                                             presort='deprecated',
                                                             random_state=42,
                                                             subsample=1.0,
                                                             tol=0.0001,
                                                             validation_fraction=0.1,
                                                             verbose=0,
                                                             warm_start=False),
                    n_jobs=None)
```

```
In [78]: prob = nb1.predict_proba(X_test)
```

```
In [84]: from itertools import cycle
p=2

plt.figure(figsize=(12,12))
colors = cycle(['aqua', 'darkorange', 'cornflowerblue', 'green', 'black'])
for i, color in zip(range(n_classes), colors):
    plt.plot(f_p_r[i], t_p_r[i], color=color, lw=p, label='ROC curve of class {0} (area = {1:0.2
f}))''.format(i, roc_auc[i]))

plt.plot([0, 1], [0, 1], 'k--', lw=p)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc="lower right")
plt.show()
```

## 6. Final predicting the test data

### TRAINING ON WHOLE DATASET AND TESTING THE UNLABELLED DATA----

```
In [98]: from sklearn.ensemble import GradientBoostingClassifier

start=timeit.default_timer()

gb2 = GradientBoostingClassifier(loss='deviance', n_estimators= 100, learning_rate= 0.1, max_de
pth = 10, random_state=42 )
gb2.fit(iso_transfer,np.ravel(y))
y_pred=gb2.predict(unlabelled_test_components)

stop=timeit.default_timer()
print("time taken",stop-start)
```

```
time taken 287.41810359100054
```

# **PCA-** **DECISION** **TREE**

## 1. Cross Val score for with different hyper parameters: {'Gini index', max\_depth}

```
from sklearn import tree
mean_accuracy=[]
timetaken=[]
max_depth=[3,5,7,10,15,20,50,75,None]
for i in max_depth:
    scores=[]
    start=timeit.default_timer()
    clfier_1=tree.DecisionTreeClassifier(max_depth=i,criterion='gini',random_state=42)
    scores=cross_val_score(clfier_1, X_train, y_train, cv=10)
    stop=timeit.default_timer()

    #splitting training data into train val data in ratio of 90:10.
    print("Mean Accuracy: ", scores.mean(), "at depth= ",i, " Time taken:", stop-start)
    mean_accuracy.append(scores.mean())
```

## 2. Plotting the accuracy vs max\_depth

```
#plotting the graph of max depth vs accuracy.
import matplotlib as plt
plt.pyplot.scatter([3,5,7,10,15,20,50,75,0],mean_accuracy)
plt.pyplot.xlabel('Max Depth')
plt.pyplot.ylabel('Mean Accuracy')
plt.pyplot.title('Max depth vs Accuracy of Decision tree classifier')
plt.pyplot.legend(loc='best', title='None relabeling', labels=['None--> 0'])
```

## 3. Cross Val score for with different hyper parameters: {'entropy', max\_depth}



```

from sklearn import tree
mean_accuracy=[]
timetaken=[]
max_depth=[3,5,7,10,15,20,50,75,None]
for i in max_depth:
    scores=[]
    start=timeit.default_timer()
    clfier_1=tree.DecisionTreeClassifier(max_depth=i,criterion='entropy',random_state
=42)
    scores=cross_val_score(clfier_1, X_train, y_train, cv=10)
    stop=timeit.default_timer()

    #splitting training data into train val data in ratio of 90:10.
    print("Mean Accuracy: ", scores.mean(),"at depth= ",i," Time taken:",stop-start)
    mean_accuracy.append(scores.mean())

```

```

#plotting the graph of max depth vs accuracy.
import matplotlib as plt
plt.pyplot.scatter([3,5,7,10,15,20,50,75,0],mean_accuracy)
plt.pyplot.xlabel('Max Depth')
plt.pyplot.ylabel('Mean Accuracy')
plt.pyplot.title('Max depth vs Accuracy of Decision tree classifier')
plt.pyplot.legend(loc='best', title='None relabeling',labels=['None--> 0'])

```

## 4. Result

### RESULT::

BEST DEPTH ⇒ 15 IN ALL PARAMETER TUNING.→ {'entropy','gini'} ¶

BEST CRITERION⇒ Entropy

Time complexity of 'entropy' is more than double to that of 'gini'.

## 5.Validation data: ‘Gini’

## APPLYING TO THE VALIDATION DATA

Accuracy: 81.38% at depth= 15 | criterion='gini' | Time taken:  
3.969800364000548

```
start=timeit.default_timer()
clfier_2=tree.DecisionTreeClassifier(max_depth=15, random_state=42)
clfier_2.fit(X_train,y_train)
stop=timeit.default_timer()
y_pred=clfier_2.predict(X_test)
acc=accuracy_score(y_pred,y_test)
print("Accuracy: ", acc,"at depth= ",15," Time taken:",stop-start)
```

```
Accuracy:  0.8138333333333333 at depth=  15  Time taken: 3.9426736169989454
```

## 6. Confusion matrix

Confusion Matrix | depth= 15 | criterion='gini'

```
In [45]: from sklearn.metrics import confusion_matrix
con_mat = confusion_matrix(y_pred, y_test)
print('Confusion Matrix:\n', con_mat)
```

```
Confusion Matrix:
[[2163  149   79    0   12]
 [ 125 1928  252   73   42]
 [   79  244 1742  256  100]
 [    8   77  223 1840  219]
 [    4   26   68  198 2093]]
```

## 7. Classification Report

## Classification Report | depth= 15 | criterion='gini'

In [46]:

```
from sklearn.metrics import classification_report
class_rep = classification_report(y_pred, y_test, target_names = ['Class-0', 'Class-1', 'Class-2', 'Class-3', 'Class-4'])
print('Classification Report:\n', class_rep)
```

```
Classification Report:
              precision    recall  f1-score   support

   Class-0       0.91      0.90      0.90      2403
   Class-1       0.80      0.80      0.80      2420
   Class-2       0.74      0.72      0.73      2421
   Class-3       0.78      0.78      0.78      2367
   Class-4       0.85      0.88      0.86      2389

 accuracy              0.81      12000
 macro avg              0.81      12000
 weighted avg           0.81      12000
```

## 8. Validation data: 'entropy'

Accuracy: 81.52% at depth= 15 | criterion='entropy' | Time taken:  
8.88460796200161

```
start=timeit.default_timer()
clfier_2=tree.DecisionTreeClassifier(max_depth=15,criterion='entropy',random_state=42)
clfier_2.fit(X_train,y_train)
stop=timeit.default_timer()
y_pred=clfier_2.predict(X_test)
acc=accuracy_score(y_pred,y_test)
print("Accuracy: ", acc,"at depth= ",20," Time taken:",stop-start)
```

Accuracy: 0.81525 at depth= 20 Time taken: 9.001226678999956

## Confusion Matrix | depth= 15 | criterion='entropy'

```
from sklearn.metrics import confusion_matrix
con_mat = confusion_matrix(y_pred, y_test)
print('Confusion Matrix:\n', con_mat)
```

```
Confusion Matrix:
[[2163  176   97    1    9]
 [ 146 1955  247   60   36]
 [  54  284 1713  278   69]
 [   2   69  237 1819  219]
 [  14   28   70  289 2133]]
```

## 9. For Roc Curve → See analysis section Criterion → 'entropy'

```
In [94]: from sklearn.tree import DecisionTreeClassifier
         nb1 = OneVsRestClassifier(DecisionTreeClassifier(max_depth=15, random_state=42, criterion='entropy'))

In [100]: prob = nb1.predict_proba(X_test)

In [108]: from itertools import cycle
         p = 2

         plt.figure(figsize=(12,12))
         colors = cycle(['aqua', 'darkorange', 'cornflowerblue', 'green', 'black'])
         for i, color in zip(range(n_classes), colors):
             plt.plot(f_p_r[i], t_p_r[i], color=color, lw=p, label='ROC curve of class {0} (area = {1:0.2f})'.format(i, roc_auc[i]))

         plt.plot([0, 1], [0, 1], 'k--', lw=p)
         plt.xlim([0.0, 1.0])
         plt.ylim([0.0, 1.05])
         plt.xlabel('False Positive Rate')
         plt.ylabel('True Positive Rate')
         plt.title('ROC Curve')
         plt.legend(loc="lower right")
         plt.show()
```

```
In [116]: nb1 = OneVsRestClassifier(tree.DecisionTreeClassifier(max_depth=15, criterion='entropy', random_state=42))
nb1.fit(X_train, y_train)
```

```
Out[116]: OneVsRestClassifier(estimator=DecisionTreeClassifier(ccp_alpha=0.0,
                                                             class_weight=None,
                                                             criterion='entropy',
                                                             max_depth=15,
                                                             max_features=None,
                                                             max_leaf_nodes=None,
                                                             min_impurity_decrease=0.0,
                                                             min_impurity_split=None,
                                                             min_samples_leaf=1,
                                                             min_samples_split=2,
                                                             min_weight_fraction_leaf=0.0,
                                                             presort='deprecated',
                                                             random_state=42,
                                                             splitter='best'),
                               n_jobs=None)
```

## 10. Final test data

### Unlabelled test data

depth = 15 , criterion = 'entropy'

```
: start=timeit.default_timer()

clfier_2=tree.DecisionTreeClassifier(max_depth=15,criterion='entropy',random_state=42)
clfier_2.fit(X_principal_components,y)
y_pred=clfier_2.predict(unlabelled_test_principal_components)

stop=timeit.default_timer()
print('Time taken',stop-start)
```

Time taken 11.650092584997765

**PCA-**  
**XGBOOST**

# 1.XG boost on validation data with varying depth.

XG BOOST ON VAL DATA || DEPTH =[5,6,8,10,20,25]

In [ ]:

```
## XGBoost
start =timeit.default_timer()
train48 = xgb.DMatrix(X_train, y_train)
val12 = xgb.DMatrix(X_test, y_test)
eval_list = [(train48, "Train"), (val12, "Validation")]

for depth in [5,6,8,10,20,25]:
    parameters_list = [("objective", "multi:softmax"), ("num_class", 5), ("max_depth", depth),
("eval_metric", "merror")]
    n_rounds = 600
    early_stopping = 40 # if val accuracy is not increasing till 40 round then it will stop and c
ontinue with next.

    XBoost = xgb.train(parameters_list, train48, n_rounds, evals=eval_list, early_stopping_rounds=early_stopping, verbose_eval=True)

    y_pred = XBoost.predict(val12)

    stop=timeit.default_timer()
    accscore=accuracy_score(y_test, y_pred)
    print('Accuracy by XGBoost: ', accscore,' Time taken', stop-start)
```

## 2. Best parameters on validation data.

### Running at best parameters for val data

Accuracy by XGBoost: 0.8934166666666666 Time taken 2272.261734103002 || Depth=20

```
In [49]: ## XGBoost
start =timeit.default_timer()
train48 = xgb.DMatrix(X_train, y_train)
val12 = xgb.DMatrix(X_test, y_test)
eval_list = [(train48, "Train"), (val12, "Validation") ]

parameters_list = [("objective", "multi:softmax"), ("num_class", 5), ("max_depth", 20),("eval_
metric", "merror")]
n_rounds = 600
early_stopping = 40 # if val accuracy is not increasing till 40 round then it will stop and conti
nue with next.

XBoost = xgb.train(parameters_list, train48, n_rounds, evals=eval_list, early_stopping_rounds=e
arly_stopping, verbose_eval=True)

y_pred = XBoost.predict(val12)

stop=timeit.default_timer()
accscore=accuracy_score(y_test, y_pred)
print('Accuracy by XGBoost: ', accscore,'at depth=20',' Time taken', stop-start)
```

### Confusion Matrix

```
In [50]: from sklearn.metrics import confusion_matrix
con_mat = confusion_matrix(y_pred, y_test)
print('Confusion Matrix:\n', con_mat)
```

```
Confusion Matrix:
[[2264   84   32    0   10]
 [  86 2146  146   44   12]
 [  26  141 2002  158   41]
 [   1   41  135 2035  129]
 [   2   12   49  130 2274]]
```



## Classification Report

```
[51]: from sklearn.metrics import classification_report
class_rep = classification_report(y_pred, y_test, target_names = ['Class-0', 'Class-1', 'Class-2', 'Class-3', 'Class-4'])
print('Classification Report:\n', class_rep)
```

```
Classification Report:
              precision    recall  f1-score   support

   Class-0       0.95      0.95      0.95     2390
   Class-1       0.89      0.88      0.88     2434
   Class-2       0.85      0.85      0.85     2368
   Class-3       0.86      0.87      0.86     2341
   Class-4       0.92      0.92      0.92     2467

 accuracy              0.89      12000
 macro avg              0.89      12000
weighted avg              0.89      12000
```

## 3. Unlabeled data prediction

### UNLABELLED TEST DATA

Final test accuracy 89.26% at depth=20

```
In [52]: start =timeit.default_timer()
test10 = xgb.DMatrix(unlabelled_test_principal_components)

y_pred1 = XBoost.predict(test10)

stop=timeit.default_timer()
print(' Time taken', stop-start)
```

```
Time taken 0.6582486719998997
```

# **PCA-** **RANDOM** **FOREST**

# 1. Cross Val score | Criterion – ‘Gini’

```
In [ ]: #applying cross val on train data .

number_of_trees=[5, 10, 50, 100, 150]
max_depth=[5,10,15,None]
mean_accuracy=[]
depth_list=[]
trees_list=[]
timetaken=[]
for no_trees in number_of_trees:
    for depth in max_depth:

        start=timeit.default_timer()

        clf_rand=RandomForestClassifier(max_depth=depth,n_estimators=no_trees, random_state=42)
        scores=cross_val_score(clf_rand, X_train, np.ravel(y_train),cv=10)

        stop=timeit.default_timer()

        print("Mean Accuracy: ", scores.mean(),"at depth= ",depth,"and when no of trees= ",no_t
rees,"Time taken ::",stop-start)
        mean_accuracy.append(scores.mean())
        depth_list.append(depth)
        trees_list.append(no_trees)
        timetaken.append(stop-start)
```

## CONVERTING THE RESULTS INTO DATAFRAME

```
In [ ]: heat_df=pd.DataFrame(data=list(zip(trees_list,depth_list,mean_accuracy,timetaken)),columns=['nu
mber_of_trees','max_depth','mean_accuracy','Time_Taken'])
```

## PLOTTING THE HEATMAP

```
In [ ]: heat_df=heat_df.pivot(index='max_depth',columns='number_of_trees',values='mean_accuracy')
sns.heatmap(data=heat_df, annot=True)
```

## 2. Cross Val score | criterion- 'entropy'

RANDOM FOREST DEFAULT PARAMETERS AND criterion='entropy'

```
In [ ]: #applying cross val on train data .

number_of_trees=[5, 10, 50, 100, 150]
max_depth=[5,10,15, None]

mean_accuracy=[]
depth_list=[]
trees_list=[]
timetaken=[]
for no_trees in number_of_trees:
    for depth in max_depth:

        start=timeit.default_timer()

        clf_rand=RandomForestClassifier(max_depth=depth,n_estimators=no_trees,criterion='entropy', random_state=42)
        scores=cross_val_score(clf_rand, X_train, np.ravel(y_train),cv=10)

        stop=timeit.default_timer()

        print("Mean Accuracy: ", scores.mean(),"at depth= ",depth,"and when no of trees= ",no_trees,"Time taken ::",stop-start)
        mean_accuracy.append(scores.mean())
        depth_list.append(depth)
        trees_list.append(no_trees)
        timetaken.append(stop-start)
```

### 3.RESULT

RESULT::----

1. NO OF TREES= 150 | DEPTH= NONE | CRITERION='gini' → 87.0
2. NO OF TREES= 150 | DEPTH= NONE | CRITERION='entropy' → 87.38
3. Entropy is better than gini in our case.
4. No of trees=150 and depth=none is giving better accuracy in both the cases. Different combination of depths and no of trees are tried.

APPLYING THE TUNED PARAMETERS i.e Depth=None and Estimators=150 AND PREDICTING THE VAL DATA (20%). 87.26% val acc , criterion='gini'

### 4. BEST parameters on validation set | ‘Gini’

APPLYING THE TUNED PARAMETERS i.e Depth=None and Estimators=150 AND PREDICTING THE VAL DATA (20%). 87.26% val acc , criterion='gini'

```
In [41]: clf_rand2 = RandomForestClassifier(max_depth=None, n_estimators=150, criterion='gini', random_state=42)
         clf_rand2.fit(X_train, np.ravel(y_train)) #Training on 48000

Out[41]: RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                                criterion='gini', max_depth=None, max_features='auto',
                                max_leaf_nodes=None, max_samples=None,
                                min_impurity_decrease=0.0, min_impurity_split=None,
                                min_samples_leaf=1, min_samples_split=2,
                                min_weight_fraction_leaf=0.0, n_estimators=150,
                                n_jobs=None, oob_score=False, random_state=42, verbose=0,
                                warm_start=False)
```

## 5. Confusion matrix

### Confusion Matrix

In [46]:

```
from sklearn.metrics import confusion_matrix
con_mat = confusion_matrix(y_pred1, y_test)
print('Confusion Matrix:\n', con_mat)
```

```
Confusion Matrix:
[[2226  86  32   0   5]
 [ 116 2102  177  41  17]
 [  36  187 1928  201  44]
 [   0   36  172 1986  170]
 [   1   13   55  139 2230]]
```

## 6. Classification report

### Classification Report

In [47]:

```
from sklearn.metrics import classification_report
class_rep = classification_report(y_pred1, y_test, target_names = ['Class-0', 'Class-1', 'Class-2', 'Class-3', 'Class-4'])
print('Classification Report:\n', class_rep)
```

```
Classification Report:
              precision    recall  f1-score   support

   Class-0       0.94      0.95      0.94       2349
   Class-1       0.87      0.86      0.86       2453
   Class-2       0.82      0.80      0.81       2396
   Class-3       0.84      0.84      0.84       2364
   Class-4       0.90      0.91      0.91       2438

 accuracy              0.87              12000
 macro avg           0.87      0.87      0.87       12000
weighted avg           0.87      0.87      0.87       12000
```

## 7. FOR ROC curve → See analysis

### ROC Curve for gini

```
In [48]: from sklearn.metrics import roc_curve, auc
         from sklearn.preprocessing import label_binarize
         from sklearn.multiclass import OneVsRestClassifier
         from sklearn.metrics import roc_auc_score
         y_labelized= label_binarize(y_test, classes=[0,1,2,3,4])

In [53]: nb1= OneVsRestClassifier(RandomForestClassifier(max_depth=None, n_estimators=150, criterion='gini', random_state=42))
         nb1.fit(X_train, y_train)

Out[53]: OneVsRestClassifier(estimator=RandomForestClassifier(bootstrap=True,
                                                                ccp_alpha=0.0,
                                                                class_weight=None,
                                                                criterion='gini',
                                                                max_depth=None,
                                                                max_features='auto',
                                                                max_leaf_nodes=None,
                                                                max_samples=None,
                                                                min_impurity_decrease=0.0,
                                                                min_impurity_split=None,
                                                                min_samples_leaf=1,
                                                                min_samples_split=2,
                                                                min_weight_fraction_leaf=0.0,
                                                                n_estimators=150,
                                                                n_jobs=None,
                                                                oob_score=False,
                                                                random_state=42, verbose=0,
                                                                warm_start=False),
                                                                n_jobs=None)

In [54]: prob = nb1.predict_proba(X_test)
```

## 8. BEST parameters on validation set | 'entropy'

APPLYING THE TUNED PARAMETERS i.e Depth=None and Estimators=150 AND PREDICTING THE VAL DATA (20%). 87.40% val acc , criterion='entropy'

```
In [61]: clf_rand2 = RandomForestClassifier(max_depth=None, n_estimators=150, criterion='entropy', random_state=42)
         clf_rand2.fit(X_train, np.ravel(y_train)) #Training on 48000
```

```
Out[61]: RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                                criterion='entropy', max_depth=None, max_features='auto',
                                max_leaf_nodes=None, max_samples=None,
                                min_impurity_decrease=0.0, min_impurity_split=None,
                                min_samples_leaf=1, min_samples_split=2,
                                min_weight_fraction_leaf=0.0, n_estimators=150,
                                n_jobs=None, oob_score=False, random_state=42, verbose=0,
                                warm_start=False)
```

### Confusion Matrix for entropy

```
[65]: from sklearn.metrics import confusion_matrix
       con_mat = confusion_matrix(y_pred1, y_test)
       print('Confusion Matrix:\n', con_mat)
```

```
Confusion Matrix:
[[2228  90  38   0   5]
 [ 120 2111  180  45  19]
 [  31  173 1918  178  43]
 [   0   34  170 2005  172]
 [   0   16   58  139 2227]]
```



## Classification Report for entropy

In [66]:

```
from sklearn.metrics import classification_report
class_rep = classification_report(y_pred1, y_test, target_names = ['Class-0', 'Class-1', 'Class-2', 'Class-3', 'Class-4'])
print('Classification Report:\n', class_rep)
```

Classification Report:				
	precision	recall	f1-score	support
Class-0	0.94	0.94	0.94	2361
Class-1	0.87	0.85	0.86	2475
Class-2	0.81	0.82	0.81	2343
Class-3	0.85	0.84	0.84	2381
Class-4	0.90	0.91	0.91	2440
accuracy			0.87	12000
macro avg	0.87	0.87	0.87	12000
weighted avg	0.87	0.87	0.87	12000

In [72]:

```
nb1= OneVsRestClassifier(RandomForestClassifier(max_depth=None, n_estimators=150, criterion='entropy', random_state=42))
nb1.fit(X_train, y_train)
```

Out[72]:

```
OneVsRestClassifier(estimator=RandomForestClassifier(bootstrap=True,
                                                    ccp_alpha=0.0,
                                                    class_weight=None,
                                                    criterion='entropy',
                                                    max_depth=None,
                                                    max_features='auto',
                                                    max_leaf_nodes=None,
                                                    max_samples=None,
                                                    min_impurity_decrease=0.0,
                                                    min_impurity_split=None,
                                                    min_samples_leaf=1,
                                                    min_samples_split=2,
                                                    min_weight_fraction_leaf=0.0,
                                                    n_estimators=150,
                                                    n_jobs=None,
                                                    oob_score=False,
                                                    random_state=42, verbose=0,
                                                    warm_start=False),
                    n_jobs=None)
```

## 9. Predicting unlabeled data

TRAINING ON WHOLE DATASET AND TESTING THE UNLABELLED DATA--87.68%, Criteria='entropy'

In [81]:

```
start=timeit.default_timer()

clf_rand3 = RandomForestClassifier(max_depth=None,n_estimators=150,criterion='entropy',random_s
tate=42)
clf_rand3.fit(X_principal_components,np.ravel(y))
y_pred2=clf_rand3.predict(unlabelled_test_principal_components)

stop=timeit.default_timer()
print("Time Taken ",stop-start)
```

Time Taken 197.69202909100022

# PCA- KNN

# 1.CROSS VAL SCORE

## CROSS VAL SCORE

### KNN | Weights='distance' | p=2(Euclidean)

```
In [ ]:
from sklearn.neighbors import KNeighborsClassifier
list=[2,5,10,15,20,25,35,50,75,100]
for k in list:
    scores=[]
    start=timeit.default_timer()

    knn_q2 = KNeighborsClassifier(n_neighbors=k,weights='distance',p=2)
    scores = cross_val_score(knn_q2, X_principal_components,y, cv=5, n_jobs = -1)
    stop=timeit.default_timer()
    print(" Mean accuracy at k=",k,"is",scores.mean(),"Time Taken",stop-start)
```

### KNN | Weights='distance' | p=1 (manhattan)

```
In [ ]:
from sklearn.neighbors import KNeighborsClassifier
list=[100]
for k in list:
    scores=[]
    start=timeit.default_timer()

    knn_q2 = KNeighborsClassifier(n_neighbors=k,weights='distance',p=1)
    scores = cross_val_score(knn_q2, X_principal_components,y, cv=5, n_jobs = -1)
    stop=timeit.default_timer()
    print(" Mean accuracy at k=",k,"is",scores.mean(),"Time Taken",stop-start)
```

## KNN | Weights='uniform' | p=2 (Euclidean)

```
In [ ]:
from sklearn.neighbors import KNeighborsClassifier
list=[2, 5, 10, 15, 20, 25, 35, 50, 75, 100]
for k in list:
    scores=[]
    start=timeit.default_timer()

    knn_q2 = KNeighborsClassifier(n_neighbors=k, weights='uniform', p=2)
    scores = cross_val_score(knn_q2, X_principal_components, y, cv=5, n_jobs = -1)
    stop=timeit.default_timer()
    print(" Mean accuracy at k=", k, "is", scores.mean(), "Time Taken", stop-start)
```

## KNN | Weights='uniform' | p=1 (manhattan)

```
In [ ]:
from sklearn.neighbors import KNeighborsClassifier
list=[2, 5, 10, 15, 20, 25, 35, 50, 75, 100]
for k in list:
    scores=[]
    start=timeit.default_timer()

    knn_q2 = KNeighborsClassifier(n_neighbors=k, weights='uniform', p=1)
    scores = cross_val_score(knn_q2, X_principal_components, y, cv=5, n_jobs = -1)
    stop=timeit.default_timer()
    print(" Mean accuracy at k=", k, "is", scores.mean(), "Time Taken", stop-start)
```

## 2. Applying best parameters on validation data | k=15 | weights= 'distance' | p=2

## TESTING ON VALIDATION DATA

Mean accuracy at k= 15 is 0.86375 Time Taken 9.94307229900005

```
1]: from sklearn.neighbors import KNeighborsClassifier

start=timeit.default_timer()

knn_q3 = KNeighborsClassifier(n_neighbors=15,weights='distance',p=2)
knn_q3.fit(X_train,np.ravel(y_train))
y_pred=knn_q3.predict(X_test)
stop=timeit.default_timer()
accscore=accuracy_score(y_test,y_pred)
print(" Mean accuracy at k=",15,"is",accscore,"Time Taken",stop-start)
```

Mean accuracy at k= 15 is 0.86375 Time Taken 8.941705003999232

## Confusion Matrix

```
2]: from sklearn.metrics import confusion_matrix
con_mat = confusion_matrix(y_pred, y_test)
print('Confusion Matrix:\n', con_mat)
```

```
Confusion Matrix:
[[2260 155 69 1 10]
 [ 101 2048 188 44 13]
 [ 16 161 1865 204 37]
 [ 1 54 193 1977 191]
 [ 1 6 49 141 2215]]
```

## Classification Report

In [43]:

```
from sklearn.metrics import classification_report
class_rep = classification_report(y_pred, y_test, target_names = ['Class-0', 'Class-1', 'Class-2', 'Class-3', 'Class-4'])
print('Classification Report:\n', class_rep)
```

```
Classification Report:
              precision    recall  f1-score   support

   Class-0       0.95      0.91      0.93     2495
   Class-1       0.84      0.86      0.85     2394
   Class-2       0.79      0.82      0.80     2283
   Class-3       0.84      0.82      0.83     2416
   Class-4       0.90      0.92      0.91     2412

 accuracy              0.86              12000
 macro avg           0.86      0.86      0.86     12000
 weighted avg       0.86      0.86      0.86     12000
```

## 3.FOR ROC → See analysis

### ROC Curve

In [51]:

```
from sklearn.metrics import roc_curve, auc
from sklearn.preprocessing import label_binarize
from sklearn.multiclass import OneVsRestClassifier
from sklearn.metrics import roc_auc_score
y_labeled= label_binarize(y_test, classes=[0,1,2,3,4])
```

In [55]:

```
knn5= OneVsRestClassifier(KNeighborsClassifier(n_neighbors=15,weights='distance',p=2))
knn5.fit(X_train, y_train)
```

Out[55]:

```
OneVsRestClassifier(estimator=KNeighborsClassifier(algorithm='auto',
                                                  leaf_size=30,
                                                  metric='minkowski',
                                                  metric_params=None,
                                                  n_jobs=None, n_neighbors=15,
                                                  p=2, weights='distance'),
                    n_jobs=None)
```

```
In [56]:  
prob = knn5.predict_proba(X_test)
```

## 4. Predicting unlabeled data

-- Unlabelled Test Data predictions

```
]:  
start=timeit.default_timer()  
knn6 = KNeighborsClassifier(n_neighbors=15,weights='distance',p=2)  
knn6.fit(X_principal_components, np.ravel(y))  
y_pred=knn6.predict(unlabelled_test_principal_components)  
stop=timeit.default_timer()  
print('Time Taken',stop-start)
```

```
Time Taken 9.522803403997386
```



**PCA- NAÏVE**  
**BAYES**

# 1. Applying the NB algorithm on validation data

```
In [45]: from sklearn.naive_bayes import GaussianNB
start=timeit.default_timer()
nb1 = GaussianNB(priors=None, var_smoothing=1e-09)
nb1.fit(X_train, np.ravel(y_train))
y_pred=nb1.predict(X_test)
stop=timeit.default_timer()
accscore_svm = accuracy_score(y_test,y_pred)
print('Time Taken',stop-start)
```

```
Time Taken 0.056841737001377624
```

## 2. Confusion matrix

### Confusion Matrix

```
In [49]: from sklearn.metrics import confusion_matrix
con_mat = confusion_matrix(y_pred, y_test)
print('Confusion Matrix:\n', con_mat)
```

```
Confusion Matrix:
[[1606  185   59   15   53]
 [ 695 1666  680  142   92]
 [  57  414 1044  413   58]
 [   2   89  432 1437  566]
 [  19   70  149  360 1697]]
```

## 3. Classification Report

### Classification Report

```
In [50]: from sklearn.metrics import classification_report
class_rep = classification_report(y_pred, y_test, target_names = ['Class-0', 'Class-1', 'Class-2', 'Class-3', 'Class-4'])
print('Classification Report:\n', class_rep)
```

```
Classification Report:
              precision    recall  f1-score   support

   Class-0       0.68      0.84      0.75      1918
   Class-1       0.69      0.51      0.58      3275
   Class-2       0.44      0.53      0.48      1986
   Class-3       0.61      0.57      0.59      2526
   Class-4       0.69      0.74      0.71      2295

 accuracy              0.62      12000
 macro avg              0.62      0.64      0.62      12000
 weighted avg           0.63      0.62      0.62      12000
```

## 4. ROC curve

### ROC Curve

```
In [51]: from sklearn.metrics import roc_curve, auc
from sklearn.preprocessing import label_binarize
from sklearn.multiclass import OneVsRestClassifier
from sklearn.metrics import roc_auc_score
y_labelized = label_binarize(y_test, classes=[0,1,2,3,4])
```

```
In [52]: y_labelized
```

```
Out[52]: array([[0, 1, 0, 0, 0],
        [1, 0, 0, 0, 0],
        [0, 0, 1, 0, 0],
        ...,
        [0, 0, 0, 0, 1],
        [0, 1, 0, 0, 0],
        [0, 1, 0, 0, 0]])
```

```
In [57]: nb1= OneVsRestClassifier(GaussianNB(priors=None, var_smoothing=1e-09))
nb1.fit(X_train, y_train)
```

```
Out[57]: OneVsRestClassifier(estimator=GaussianNB(priors=None, var_smoothing=1e-09),
                             n_jobs=None)
```

```
In [60]: prob = nb1.predict_proba(X_test)
```

## 5. Predicting unlabeled data

-- Unlabelled Test Data predictions

```
In [67]: start=timeit.default_timer()
nb2 = GaussianNB(priors=None, var_smoothing=1e-09)
nb2.fit(X_principal_components, np.ravel(y))
y_pred=nb2.predict(unlabelled_test_principal_components)
stop=timeit.default_timer()
print('Time Taken', stop-start)
```

```
Time Taken 0.0559105990032549
```

# **PCA- Logistic** **Regression**

# 1. Applying logistic regression on validation data

Logistic regression with default parameters resulted in accuracy of 69% on validation data.

```
In [41]: # all parameters not specified are set to their defaults
from sklearn.linear_model import LogisticRegression
start=timeit.default_timer()
logisticRegr = LogisticRegression(solver = 'lbfgs',max_iter=4000)
logisticRegr.fit(X_train, np.ravel(y_train))
y_pred=logisticRegr.predict(X_test)
stop=timeit.default_timer()
print('Time Taken',stop-start)
```

```
Time Taken 7.111053730000094
```

## 2. Confusion matrix

### Confusion Matrix

```
In [46]: from sklearn.metrics import confusion_matrix
con_mat = confusion_matrix(y_pred, y_test)
print('Confusion Matrix:\n', con_mat)
```

```
Confusion Matrix:
[[1935  321   53    0   15]
 [ 430 1511  519   26   83]
 [   13  528 1292  339   58]
 [    1   32  400 1683  394]
 [    0   32  100  319 1916]]
```

## 3. Classification Report

### Classification Report

```
In [47]: from sklearn.metrics import classification_report
class_rep = classification_report(y_pred, y_test, target_names = ['Class-0', 'Class-1', 'Class-2', 'Class-3', 'Class-4'])
print('Classification Report:\n', class_rep)
```

```
Classification Report:
              precision    recall  f1-score   support

   Class-0       0.81      0.83      0.82     2324
   Class-1       0.62      0.59      0.61     2569
   Class-2       0.55      0.58      0.56     2230
   Class-3       0.71      0.67      0.69     2510
   Class-4       0.78      0.81      0.79     2367

 accuracy              0.69      12000
 macro avg              0.69      12000
 weighted avg           0.69      12000
```

## 4. ROC curve

### ROC Curve

```
In [48]: from sklearn.metrics import roc_curve, auc
from sklearn.preprocessing import label_binarize
from sklearn.multiclass import OneVsRestClassifier
from sklearn.metrics import roc_auc_score
y_labelized= label_binarize(y_test, classes=[0,1,2,3,4])
```

```
In [52]: log1 = OneVsRestClassifier(LogisticRegression(solver = 'lbfgs', max_iter=4000))
log1.fit(X_train, y_train)

Out[52]: OneVsRestClassifier(estimator=LogisticRegression(C=1.0, class_weight=None,
dual=False, fit_intercept=True,
intercept_scaling=1,
l1_ratio=None, max_iter=4000,
multi_class='auto',
n_jobs=None, penalty='l2',
random_state=None,
solver='lbfgs', tol=0.0001,
verbose=0, warm_start=False),
n_jobs=None)
```

```
In [54]: prob = log1.decision_function(X_test)
```

## 5. Predicting unlabeled test data

-- Unlabelled Test Data predictions

```
In [61]: start=timeit.default_timer()
logisticRegr = LogisticRegression(solver = 'lbfgs', max_iter=4000)
logisticRegr.fit(X_principal_components, np.ravel(y))
y_pred=logisticRegr.predict(unlabelled_test_principal_components)
stop=timeit.default_timer()
print('Time Taken', stop-start)
```

```
Time Taken 9.360347406000074
```



## 1.3: Kaggle Competition Score (5 points)

1. Highest accuracy reached on Kaggle is 90.06% at  $c=11.7$  in SVM after taking 42 components using PCA as 42 components explained the total variance of 85% which we think is a decent cutoff point for reducing the number of features (From 784) which doesn't add up anything in classification.
2. When total explained variance to be considered was taken as 95%, number of components chosen by it were 187. When final prediction was made using the SVM with  $C=11.7$ , final accuracy on unseen data was 89.60%.
3. When XGBOOST was applied on final test data with tuned parameters taking  $\text{max\_depth}=20$ , accuracy was 89.26%.

### Note→

1. Our focus in the competition was to get the high-level idea of which algorithms work on our dataset with tweaking the most effective hyperparameters. We have taken into consideration most effective hyperparameters with multiple values and combinations and validated the same with `randomsearchcv`, `crossvalscore` etc.
2. Other teams might have focused on few algorithms and tweaked multiple hyperparameters to get the highest accuracy.

(1.4)

# **RESULT ANALYSIS**

## **POINT OF FOCUS: TIME COMPLEXITY**

### **TIME COMPLEXITY WITHIN THE ALGORITHMS WITH CHANGING PARAMETERS**

#### **1.PCA-SVM**

**BEST HYPERPARAMETERS CHOSEN USING CROSS VAL  
SCORE. FOR ALL HYPERPARAMETER'S TIME AND  
ACCURACY PLEASE CONSULT CODE.**

**1. Hyperparameters: {C=10, kernel='rbf', degree=3}**

**Time: 81.92**

**Accuracy: 89.64%**

**2. Hyperparameters: {C=11.7, kernel='rbf', degree=3}**

**Time: 82.24**

**Accuracy: 89.69%**

**3. Hyperparameters: {C=12, kernel='rbf', degree=3}**

**Time: 81.92**

**Accuracy: 89.70%**

**4. Hyperparameters: {C=10, kernel='poly'}**

**Time: 95.32**  
**Accuracy: 87.64%**

## ANALYSIS:

1. Time for the model to converge is not changing much with the changing value of C which can be seen in the above tuned parameters.
2. Thus, without considering time, we need to consider the case of overfitting and accuracy.
3. Here we can simply choose the value of  $C=11.7/12$  as it is converging well, and the is almost similar for other values as well.
4. Note: C was chosen by random search cv and cross Val score.

## 2.PCA-Random Forest

**BEST HYPERPARAMETERS CHOSEN USING CROSS VAL SCORE. FOR ALL HYPERPARAMETER'S TIME AND ACCURACY PLEASE CONSULT CODE.**

1. Hyperparameters:  
`{depth=None,n_estimators=150,criterion='gini' }`

**Time: 63.72**  
**Accuracy: 87.26%**

**2. Hyperparameters:{depth=None,n\_estimators=150,  
criterion='entropy'}**

**Time: 152.843**

**Accuracy: 87.40%**

## **ANALYSIS:**

**1. It can be clearly seen that the time complexity when the criterion was 'Gini' was decent and model took 63 seconds for training and testing whereas 'Entropy' took more than double to the time taken by the former.**

**2. Accuracy was improved in 'Entropy'.**

**Accuracy change=  $87.40 - 87.26 = 0.14$  % which is not a great improvement when time is considered.**

**3. Thus, considering time, we need to take the Gini index as the parameter for criterion.**

## **REASON:**

**→ Entropy:** Information gain finds the difference between entropy before split and average entropy after split of the dataset based on given feature values. Thus, taking longer time for computation.

**→ Gini Index:** The Gini Index performs the binary split for each feature.

## **3.PCA-Decision Tree**

**BEST HYPERPARAMETERS CHOSEN USING CROSS VAL SCORE. FOR ALL HYPERPARAMETER'S TIME AND ACCURACY PLEASE CONSULT CODE.**

**1. Hyperparameters: {depth=15, criterion='Gini'}**

**Time: 3.96**

**Accuracy: 81.38%**

**2. Hyperparameters: {depth=15, criterion='entropy'}**

**Time: 8.8846**

**Accuracy: 81.52%**

### **ANALYSIS:**

- 1. Time complexity for decision Tree with criterion='Gini' is less than half to that of the 'entropy'.**
- 2. Again, if we consider the time complexity, Criterion='Gini' is to be chosen because accuracy is not affected much.**

### **REASON:**

- Entropy:** Information gain finds the difference between entropy before split and average entropy after split of the dataset based on given feature values. Thus, taking longer time for computation.
- Gini Index:** The Gini Index performs the binary split for each feature.

## 4.LDA-SVM

**BEST HYPERPARAMETERS CHOSEN USING CROSS VAL SCORE. FOR ALL HYPERPARAMETER'S TIME AND ACCURACY PLEASE CONSULT CODE.**

1. Hyperparameters: {C=10, kernel='rbf', degree=3}

**Time: 65.208**

**Accuracy: 72.76%**

### ANALYSIS:

1. Time taken by the LDA-SVM for the same hyperparameters used for PCA-SVM is less by 20 seconds and the accuracy is extremely traded off.
2. SVM is faster for LDA because the number of features extracted by LDA = (Number of class labels)-1 irrespective of the explained variance. As it focuses on maximizing the separation between classes while minimizing the variance within the class resulting in len(class)-1 labels.

## 5.PCA-Naïve Bayes

**BEST HYPERPARAMETERS CHOSEN USING CROSS VAL SCORE. FOR ALL HYPERPARAMETER'S TIME AND ACCURACY PLEASE CONSULT CODE.**

**1. Hyperparameters: {prior=None, var\_smoothing = 1e-09}**

**Time: 0.05684**

**Accuracy: 62.083%**

#### **ANALYSIS:**

- 1. Time taken by the Naïve Bayes is extremely less because of the fact that it has the trivial operations of counting and dividing which makes the computations fast.**
- 2. It only requires the prior probability values which are constant, and the same values are used.**
- 3. As discussed in part (1.1), We chose Naïve Bayes to analyze the time and use it as the baseline for the linear classifiers.**
- 4. Accuracy of 62% can be considered as worst but it is 4 times of random guessing for our dataset.**

## **6.PCA-Logistic Regression**



**BEST HYPERPARAMETERS CHOSEN USING CROSS VAL SCORE. FOR ALL HYPERPARAMETER'S TIME AND ACCURACY PLEASE CONSULT CODE.**

**1. Hyperparameters: {solver= lbfgs, max\_iter=4000}**

**Time: 7.111**

**Accuracy: 69.475%**

### **ANALYSIS:**

- 1. Time taken by the Logistic regression is 7 seconds which is similar to the Naïve Bayes with a slight improvement in the accuracy.**
- 2. Logistic regression being strictly convex makes it faster even for the larger datasets.**
- 3. Decision surface of logistic regression is linear, in order to compare how the linear classifiers, generalize on our Fashion MNSIT dataset and compare its performance with the Naïve Bayes, we tried the logistic regression.**

## **7.PCA-Gradient Boosting**

**BEST HYPERPARAMETERS CHOSEN USING CROSS VAL SCORE. FOR ALL HYPERPARAMETER'S TIME AND ACCURACY PLEASE CONSULT CODE.**

**1. Hyperparameters:**

**{max\_depth=10, n\_estimators=100, lr=0.1, loss='deviance', criterion='gini'}**

**Time: 1422**

**Accuracy: 88.275%**

**ANALYSIS:**

**1. Time complexity for gradient boosting for the above hyperparameters is about 1422 seconds which is much higher because of the fact that it keeps on minimizing the error by forming 1 tree at a time and it keeps on adjusting the shortcomings of existing weak learner.**

**2. In cross Val score, Time complexity for Gini and entropy is given below:**

**a. {max\_depth=10, n\_estimators=10, lr=0.01, loss='deviance', criterion='gini'} → Time: 752.5**

**b. {max\_depth=30, n\_estimators=10, lr=0.01, loss='deviance', criterion='gini'} → Time: 1366.96**

**Result: Above two results are taken from our code of cross val score, which shows changing the depth has a major effect on time.**

**Various other parameters were tried, and it was found that:**

1. **Max\_depth, n\_estimators are directly proportional to the time complexity.**

**Ex:**

- a. Mean Accuracy at learning rate=0.01, n\_estimators= 10 and at max\_depth=10 is 82.34% where Time taken is 752.
- b. Mean Accuracy at learning rate=0.01, n\_estimators= 50 and at max\_depth=10 is 84.33% where Time taken is 3498.92

Thus,  $(n\_estimators * 5) \rightarrow (Time * 5)$

## **8.LDA-Gradient Boosting**

**BEST HYPERPARAMETERS CHOSEN USING CROSS VAL SCORE. FOR ALL HYPERPARAMETER'S TIME AND ACCURACY PLEASE CONSULT CODE.**

1. **Hyperparameters:**

**{max\_depth=10, n\_estimators=100, lr=0.1, loss='deviance', criterion='gini'}**

**Time: 161.147**

**Accuracy: 71.40%**

### **ANALYSIS:**

1. **Time taken by the LDA-Gradient Boosting for the same hyperparameters used for PCA-Gradient Boosting is extremely less and the accuracy is extremely traded off.**
2. **GB is faster for LDA because the number of features extracted by LDA = (Number of class labels)-1.**

## 9.PCA-KNN

**BEST HYPERPARAMETERS CHOSEN USING CROSS VAL SCORE. FOR ALL HYPERPARAMETER'S TIME AND ACCURACY PLEASE CONSULT CODE.**

### 1. Hyperparameters:

{k=15, weights='distance', p=2}

Time: 9.94

Accuracy: 86.375%

### 2. CROSS VAL SCORE:

Let's take the value of k=2 from our code only. We have tested for multiple k values.

#### a. KNN | Weights='distance' | p=2(Euclidean)

Mean accuracy at k= 2 is 0.8335333333333332 Time Taken 20.9283822

#### b. KNN | Weights='distance' | p=1 (manhattan)

Mean accuracy at k= 2 is 0.8179833333333333 Time Taken 78.98695889599912

#### c. KNN | Weights='uniform' | p=2 (Euclidean)

Mean accuracy at k= 2 is 0.8177 Time Taken 24.549888894995092

## **d. KNN | Weights='uniform' | p=1 (manhattan)**

Mean accuracy at k= 2 is 0.7990999999999999 Time Taken 81.6868522060031

### **ANALYSIS:**

1. It can be clearly seen from the above results that when the weights = [distance or uniform] and p is Euclidean i.e. p=2, time taken is less than 4 times the time taken when p=1.
2. When the weights are to be considered the time taken by the model is almost similar.

## **10. PCA-XGBoost**

**BEST HYPERPARAMETERS CHOSEN USING CROSS VAL SCORE. FOR ALL HYPERPARAMETER'S TIME AND ACCURACY PLEASE CONSULT CODE.**

1. Hyperparameters:  
{objective: multi: softmax, max\_depth=5}

**Time: 333.67**

**Accuracy: 88.13%**

2. Hyperparameters:  
{objective: multi: softmax, max\_depth=6}

**Time: 891.22**

**Accuracy: 89.02%**

**3. Hyperparameters:**

**{objective: multi: softmax, max\_depth=8}**

**Time: 1351.35**

**Accuracy: 88.99%**

**4. Hyperparameters:**

**{objective: multi: softmax, max\_depth=10}**

**Time: 1788.06**

**Accuracy: 89.24%**

**5. Hyperparameters:**

**{objective: multi: softmax, max\_depth=20}**

**Time: 2272.26**

**Accuracy: 89.34%**

**6. Hyperparameters:**

**{objective: multi: softmax, depth=25}**

**Time: 2685.08**

**Accuracy: 89.18%**

**ANALYSIS:**

1. With the increase in the max\_depth, the time is increasing exponentially in the beginning.
2. If we take time into consideration, then depth =5 is the most appealing value for depth with a tradeoff of about 1.21% of validation accuracy which is not a good choice.

## TIME COMPLEXITY BETWEEN THE ALGOS:

Taking the best hyperparameters considering the time complexity as well as accuracy. Following are the results:

Classifiers	Time	Accuracy(%)
PCA-SVM	82.24	89.69
PCA-RF	63.72	87.26
PCA-DT	3.96	81.38
LDA-SVM	65	72.76
PCA-NB	0.056	62.08
PCA-LR	7.11	69.475
PCA-GB	1422	88.275
LDA-GB	161	71.40
PCA-KNN	9.94	86.375
PCA-XGB	333	88.13

### Analysis:

1. If we are extremely focused on time, then Decision tree can be a good choice for our dataset.
2. If we are focused on both the accuracy and time, then PCA-SVM is a great choice given the accuracy and the time.

# POINT OF FOCUS: ACCURACY

## COMPARISON OF ALGORITHM AND PARAM. WITHIN THE ALGO AND BETWEEN THE ALGOS.

### 1.PCA-SVM

#### 1. Hyperparameters → {kernel, C, degree}

**C: (Default: 1)** It is the regularization parameter which is responsible for understanding how much misclassification can be afforded by each training example. It is inversely proportional to the margin between the classes in the hyperplane. Large value of c leads to overfitting the model.

**Kernel: (Default: 'rbf')** The hyperplane decision boundary between the classes is defined by the kernels. Poly kernel generates new features by using the polynomial combination of the extracted features using PCA components (in our case) whereas 'radial basis function' does the same by finding the distance between a data instance and all the other instances.

**Degree: (Default: 3)** Ignored by all the other kernels except poly.

#### 2. Usage scenario/Design choice: (Number of parameters)

```
c_list=[0.1,0.5,1,2,3,5,10,11,12,15,20,30]
```

1. We have used kernel= {'rbf', 'poly'} and cross validated on above c values. (Also performed Randomized search on loguniform scale.



2. Best accuracy was found at  $C=11.7$  and  $\text{kernel} = \text{'rbf'}$  for which we validated our model on 12000 values (20% of training data).
3. **Design choice** for our problem of image classification with SVM is
  - PCA with 85% total explained variance.(42 components)
  - $\text{SVC}(C=11.7, \text{kernel}=\text{'rbf'})$  with no overfitting and low variance.
  - For Complete design, refer page 1.

### 3. Confusion matrix:

For  $\{C= 11.7, \text{kernel} = \text{'rbf'}\}$  on Validation set:  
Accuracy → 89.69%

```
Confusion Matrix:
[[2271  107   28    0    3]
 [  90 2133  163   44   11]
 [  18  145 1992  150   31]
 [   0   33  146 2066  120]
 [   0    6   35  107 2301]]
```

## 4. Classification Report:

Classification Report:				
	precision	recall	f1-score	support
Class-0	0.95	0.94	0.95	2409
Class-1	0.88	0.87	0.88	2441
Class-2	0.84	0.85	0.85	2336
Class-3	0.87	0.87	0.87	2365
Class-4	0.93	0.94	0.94	2449
accuracy			0.90	12000
macro avg	0.90	0.90	0.90	12000
weighted avg	0.90	0.90	0.90	12000

### Analysis:

1. SVM applied with best parameters on principle components classify very well and the best classified class label is class label 0 with a precision of 0.95.
2. Highly misclassified class label is class label 2 with precision of 0.84.
3. Recall rate of class is 0 is highest.
4. Thus, SVM with best chosen parameters classifies class 0 with high accuracy.

## 2. PCA-RANDOM FOREST

### 1. hyperparameters: {n\_estimators, criterion, max\_depth,}

**n\_estimators** (Default: 100)→ Total number of trees in the forest. With increase in the n\_estimators, the time complexity increases drastically but the accuracy also improves in this trade off.

**Criterion** (Default: 'Gini')→ Measurement of the quality of the split. It can also be 'entropy' for the information gain whereas 'Gini' is used for the 'Gini Index'.

**Max\_depth** (Default: None)→ Maximum depth of the tree till the last leaf node is achieved.

### 2. Usage scenario/Design choice: (Number of parameters)

```
number_of_trees=[5, 10, 50, 100, 150]  
max_depth=[5, 10, 15, None]
```

1. We have used **Criterion= {'gini', 'entropy'}** with the above parameters and cross validated.
2. Best accuracy was found at n\_estimators=150 and criterion= 'entropy' for which we validated our model on 12000 values (20% of training data).

RESULT::----

1. NO OF TREES= 150 | DEPTH= NONE | CRITERION='gini' → 87.0

2. NO OF TREES= 150 | DEPTH= NONE | CRITERION='entropy' → 87.38

3. Entropy is better than gini in our case.

4. No of trees=150 and depth=none is giving better accuracy in both the cases. Different combination of depths and no of trees are tried.

3. **Design choice** for our problem of image classification with RF is

- PCA with 85% total explained variance.(42 components)
- RandomForestClassifier(max\_depth=None,n\_estimators=150,Criterion='entropy',random\_state=42).
- For Complete design, refer page 1.

### 3. Confusion matrix: {'entropy'}

For{max\_depth=None,  
n\_estimators=150,Criterion='entropy'} on Validation set:  
Accuracy → 87.40%

Confusion Matrix:

```
[[2228   90   38    0    5]
 [ 120 2111  180   45   19]
 [   31  173 1918  178   43]
 [    0   34  170 2005  172]
 [    0   16   58  139 2227]]
```

#### 4. Classification Report: {'entropy'}

Classification Report:				
	precision	recall	f1-score	support
Class-0	0.94	0.94	0.94	2361
Class-1	0.87	0.85	0.86	2475
Class-2	0.81	0.82	0.81	2343
Class-3	0.85	0.84	0.84	2381
Class-4	0.90	0.91	0.91	2440
accuracy			0.87	12000
macro avg	0.87	0.87	0.87	12000
weighted avg	0.87	0.87	0.87	12000

For{max\_depth=None, n\_estimators=150,Criterion='gini' }  
on Validation set:  
Accuracy → 87.26%

Confusion Matrix:					
[	2226	86	32	0	5]
[	116	2102	177	41	17]
[	36	187	1928	201	44]
[	0	36	172	1986	170]
[	1	13	55	139	2230]]

**Gini**

Classification Report:				
	precision	recall	f1-score	support
Class-0	0.94	0.95	0.94	2349
Class-1	0.87	0.86	0.86	2453
Class-2	0.82	0.80	0.81	2396
Class-3	0.84	0.84	0.84	2364
Class-4	0.90	0.91	0.91	2438
accuracy			0.87	12000
macro avg	0.87	0.87	0.87	12000
weighted avg	0.87	0.87	0.87	12000

## Gini

### Analysis: {For 'entropy'}

1. Random Forest applied with best parameters on principle components best classified class label 0 with a precision of 0.94.
2. Highly misclassified class label is class label 2 with precision of 0.81.
3. Recall rate of class is 0 is highest followed by class 4.
4. Thus, RF with best chosen parameters classifies class 0 with high accuracy.

# 3.PCA-DECISION TREE

## 1. hyperparameters: {n\_estimators, criterion, max\_depth,}

**Criterion** (Default: 'Gini')→ Measurement of the quality of the split. It can also be 'entropy' for the information gain whereas 'Gini' is used for the 'Gini Index'. Entropy somehow improves the accuracy trading off with time.

**Max\_depth** (Default: None)→ Maximum depth of the tree till the last leaf node is achieved.

## 2. Usage scenario/Design choice: (Number of parameters)

```
max_depth=[3, 5, 7, 10, 15, 20, 50, 75, None]
```

1. We have used **Criterion= {'gini', 'entropy'}** with the above parameters and cross validated.
2. Best accuracy was found at **depth=15** and **criterion= 'entropy'** but the time complexity was twice as that of 'gini'. But we still chose 'entropy' if we are focused on accuracy and the time complexity is already very low making the difference negligible in practice. we validated our model on 12000 values (20% of training data).

### RESULT::

**BEST DEPTH ⇒ 15 IN ALL PARAMETER TUNING. → {'entropy','gini'}**

**BEST CRITERION ⇒ Entropy**

**Time complexity of 'entropy' is more than double to that of 'gini'.**

3. **Design choice** for our problem of image classification with DT is

- PCA with 85% total explained variance.(42 components)
- DecisionTreeClassifier(max\_depth=15,Criterion='entropy').
- For Complete design, refer page 1.

### 3. Confusion matrix: {'entropy'}

For{max\_depth=15, Criterion='entropy'} on Validation set:  
Accuracy → 81.52%

```
Confusion Matrix:
[[2163  176   97    1    9]
 [ 146 1955  247   60   36]
 [   54  204 1713  278   69]
 [    2   69  237 1819  219]
 [   14   20   70  209 2133]]
```

### Entropy

### 4. Classification Report:

```
Classification Report:
              precision    recall  f1-score   support

   Class-0       0.91       0.88       0.90       2446
   Class-1       0.81       0.80       0.80       2444
   Class-2       0.72       0.74       0.73       2318
   Class-3       0.77       0.78       0.77       2346
   Class-4       0.86       0.87       0.87       2446

 accuracy              0.82       12000
  macro avg           0.81       0.81       0.81       12000
 weighted avg           0.82       0.82       0.82       12000
```

### Entropy



For{max\_depth=15, Criterion=gini} on Validation set:  
Accuracy → 81.38%

```
Confusion Matrix:  
[[2163  149   79    0   12]  
 [ 125 1928  252   73   42]  
 [   79  244 1742  256  100]  
 [    8   77  223 1840  219]  
 [    4   26   68  198 2093]]
```

### Gini

```
Classification Report:  
                precision    recall  f1-score   support  
  
   Class-0       0.91        0.90        0.90       2403  
   Class-1       0.80        0.80        0.80       2420  
   Class-2       0.74        0.72        0.73       2421  
   Class-3       0.78        0.78        0.78       2367  
   Class-4       0.85        0.88        0.86       2389  
  
   accuracy              0.81              12000  
   macro avg           0.81        0.81        0.81       12000  
   weighted avg        0.81        0.81        0.81       12000
```

### Gini

## Analysis: {For 'entropy'}

1. Decision Tree applied with best parameters{depth=15, criterion='entropy'} on principle components best classified class label 0 with a precision of 0.91.

2. Highly misclassified class label is class label 2 with precision of 0.72.
3. Recall rate of class is {0,1} is highest followed by class 4.
4. Thus, DT with best chosen parameters classifies class 0 with high accuracy.

## 4.LDA-SVM

### 1. Hyperparameters → {kernel, C}

**C: (Default: 1)** It is the regularization parameter which is responsible for understanding how much misclassification can be afforded by each training example. It is inversely proportional to the margin between the classes in the hyperplane. Large value of c leads to overfitting the model.

**Kernel: (Default: 'rbf')** The hyperplane decision boundary between the classes is defined by the kernels. Poly kernel generates new features by using the polynomial combination of the extracted features using PCA components (in our case) whereas 'radial basis function' does the same by finding the distance between a data instance and all the other instances.

### 2. Usage scenario/Design choice: (Number of parameters)

```
c_list=[0.1,0.5,1,2,3,5,10]
```

1. We have used kernel= {'rbf'} and cross validated on above c values.

2. Best accuracy was found at C=10 and kernel= 'rbf' for which we validated our model on 12000 values (20% of training data).

3. **Design choice** for our problem of image classification with SVM is

→ LDA(solver='svd')

→ SVC(C=10, kernel='rbf') with no overfitting and low variance.

→ For Complete design, refer page 1.

#### 4. Confusion matrix:

For {C= 10, kernel= 'rbf'} on Validation set:

Accuracy → 72.76%

Confusion Matrix:

```
[[1992  264   32    0    3]
 [ 369 1726  559   43   39]
 [   15  378 1392  415   50]
 [    1   31  318 1590  342]
 [    2   25   63  319 2032]]
```

## 5. Classification Report:

Classification Report:				
	precision	recall	f1-score	support
Class-0	0.84	0.87	0.85	2291
Class-1	0.71	0.63	0.67	2736
Class-2	0.59	0.62	0.60	2250
Class-3	0.67	0.70	0.68	2282
Class-4	0.82	0.83	0.83	2441
accuracy			0.73	12000
macro avg	0.73	0.73	0.73	12000
weighted avg	0.73	0.73	0.73	12000

## Analysis:

1. SVM applied with best parameters on LDA classify class label 0 with a precision of 0.84.
2. Highly misclassified class label is class label 2 with precision of 0.59.
3. Recall rate of class is 0 is highest.
4. Thus, LDA-SVM with best chosen parameters classifies class 0 with high accuracy.
5. LDA is optimal if and only if the classes are Gaussian and have equal covariance.

# 5.PCA-Naïve Bayes

## 1. Hyperparameters→ {Priors, var\_smoothing}

**Priors: (Default: None):** Priors are the probability of the classes. If specified explicitly, they are not adjusted according to the dataset.

**Var\_smoothing(Default: 1e-9):** Area specifying largest variance of all features which is added to the variances for calculation stability.

**Note:** It makes the strong data distribution assumption that any two features are independent for a class label.

## 2. Usage scenario/Design choice: (Number of parameters)

1. We have used default values for both the parameters {'priors', var\_smoothing}.
2. We are using this linear classifier to check how the linear classifiers perform on an image dataset like ours.
3. **Design choice** for our problem of image classification with SVM is
  - PCA with 85% total explained variance.(42 components)
  - GaussianNB(priors=None, var\_smoothing=1e-09).
  - For Complete design, refer page 1.

## 3. Confusion matrix:

For { priors=None, var\_smoothing=1e-09} on Validation set:

Accuracy → 62.08%

Confusion Matrix:

```
[[1606  185   59   15   53]
 [ 695 1666  680  142   92]
 [   57  414 1044  413   58]
 [    2   89  432 1437  566]
 [   19   70  149  360 1697]]
```

#### 4. Classification Report:

Classification Report:

	precision	recall	f1-score	support
Class-0	0.68	0.84	0.75	1918
Class-1	0.69	0.51	0.58	3275
Class-2	0.44	0.53	0.48	1986
Class-3	0.61	0.57	0.59	2526
Class-4	0.69	0.74	0.71	2295
accuracy			0.62	12000
macro avg	0.62	0.64	0.62	12000
weighted avg	0.63	0.62	0.62	12000

#### Analysis:

1. Naïve Bayes classifies class 1, class 4 and class 0 equally by giving a precision of 0.68 and 0.69.

2. Highly misclassified class label is class label 2 with precision of 0.44.
3. Recall rate of class is 0 is highest.
4. Thus, Naïve Bayes with best chosen parameters classifies class 0,1,4 with almost equal precision.
5. If a condition is completely unseen by the model, then it will not correctly classify the test data for that condition as it works on the conditional probability.
6. It provided better accuracy than random guessing for all the class labels.

## 6.PCA-LOGISTIC REGRESSION

### 1. Hyperparameters → {solver, max\_iter}

**solver: (Default: lbfgs):** Algorithm for the optimization problem. For multi class problems, lbfgs is used.

**Max\_iter(Default: 100):** Number of iterations required to converge by the solver(lbfgs in our case).

**Note:** The likelihood function will stop converging once the full separation is achieved.

.

## 2. Usage scenario/Design choice: (Number of parameters)

1. Using the default values with `max_iter=4000` to let the `lbfgs` converge properly on our dataset.
2. We are using this linear classifier to check how the linear classifiers performs and compare the performance of both Naïve Bayes and logistic regression.
3. **Design choice** for our problem of image classification with SVM is
  - ➔ PCA with 85% total explained variance.(42 components)
  - ➔ `LogisticRegression(solver=lbfgs, max_iter=4000)`.
  - ➔ For Complete design, refer page 1.

## 3. Confusion matrix:

For { `solver=lbfgs, max_iter=4000` } on Validation set:  
Accuracy ➔ 69.475%

Confusion Matrix:

```
[[1935  321   53    0   15]
 [ 430 1511  519   26   83]
 [   13  528 1292  339   58]
 [    1   32  400 1683  394]
 [    0   32  100  319 1916]]
```



## 4. Classification Report:

Classification Report:				
	precision	recall	f1-score	support
Class-0	0.81	0.83	0.82	2324
Class-1	0.62	0.59	0.61	2569
Class-2	0.55	0.58	0.56	2230
Class-3	0.71	0.67	0.69	2510
Class-4	0.78	0.81	0.79	2367
accuracy			0.69	12000
macro avg	0.69	0.70	0.69	12000
weighted avg	0.69	0.69	0.69	12000

## Analysis:

1. Logistic Regression classifies class 0 very well as compared to the Naïve Bayes by giving a precision of 0.81 which is almost 1.19 times of that provided by Naïve Bayes.
2. Highly misclassified class label is again class label 2 with precision of 0.55 which is still better than that of Naïve Bayes.
3. Recall rate of class is 0 is highest followed by the class 4.

4. Thus, Logistic regression is much better than the Naïve Bayes for classifying images given the fact that it doesn't depend on the conditional probabilities.

## 7.PCA- GRADIENT BOOSTING

### 1. hyperparameters:{n\_estimators,learning\_rate,max\_depth,criterion}

**n\_estimators** (Default: 100)→ Total number of trees in the forest. With increase in the n\_estimators, the time complexity increases drastically but the accuracy also improves in this trade off.

**Max\_depth** (Default: 3)→ Maximum depth of the tree till the last leaf node is achieved.

**Learning\_rate** (Default: 0.1)→ In optimization algorithm, learning rate determines the step size at each iteration while finding the minimum of a loss function(say deviance)

### 2. Usage scenario/Design choice: (Number of parameters)

```
number_of_estimators=[5,10,25,50,100]  
learning_rate=[0.001,0.01,0.05,0.1,1]
```

1. We have used **max\_depth= {3,5,10,15,20,50}** with the above parameters and cross validated.
2. Best accuracy was found at n\_estimators=100, learning\_rate=0.1, max\_depth= 10 for which we validated our model on 12000 values (20% of training data).
3. An accuracy of 88.27% was achieved at the above hyperparameters.

## RESULT →

**BEST parameters using cross val score are:**

→ learning\_rate=0.1

→ n\_estimators= 100

→ max\_depth=10

4. **Design choice** for our problem of image classification with RF is

- PCA with 85% total explained variance.(42 components)
- GradientBoostingClassifier(loss='deviance',n\_estimators=100,learning\_rate=0.1, max\_depth=10).
- For Complete design, refer page 1.

### 3. Confusion matrix:

For{ loss='deviance',n\_estimators=100,learning\_rate=0.1, max\_depth=10} on Validation set:  
Accuracy → 88.275%

```
Confusion Matrix:
[[2249   92   41    0    9]
 [  97 2114  150   47   13]
 [  31  169 1970  183   54]
 [    1   39  154 2000  130]
 [    1   10   49  137 2260]]
```

## 4. Classification Report:

Classification Report:				
	precision	recall	f1-score	support
Class-0	0.95	0.94	0.94	2391
Class-1	0.87	0.87	0.87	2421
Class-2	0.83	0.82	0.83	2407
Class-3	0.84	0.86	0.85	2324
Class-4	0.92	0.92	0.92	2457
accuracy			0.88	12000
macro avg	0.88	0.88	0.88	12000
weighted avg	0.88	0.88	0.88	12000

## Analysis:

1. Gradient Boosting on principle components classifies the class 0 very efficiently with a precision of 0.92
2. Highly misclassified class label is again class label 2 and class label 3 with precision of 0.83 and 0.84.
3. Recall rate of class is 0 is highest followed by the class 4.

## 8.LDA-GRADIENT BOOSTING

### 1. hyperparameters:{n\_estimators,learning\_rate,max\_depth,criterion}

**n\_estimators** (Default: 100)→ Total number of trees in the forest. With increase in the n\_estimators, the time complexity increases drastically but the accuracy also improves in this trade off.

**Max\_depth** (Default: 3)→ Maximum depth of the tree till the last leaf node is achieved.

**Learning\_rate** (Default: 0.1)→ In optimization algorithm, learning rate determines the step size at each iteration while finding the minimum of a loss function(say deviance)

### 2. Usage scenario/Design choice: (Number of parameters)

```
lr = [0.1,1]
n_estimators = [100]
max_depth = [10]
```

1. We have used **max\_depth= {10}** with the above parameters and tested on validation set.

## RESULT---

1. Mean Accuracy at learning rate= 0.1 , n\_estimators= 100 and at max\_depth= 10 is 0.7140833333333333 where Timetaken is 156.34929820499383

2. Mean Accuracy at learning rate= 1 , n\_estimators= 100 and at max\_depth= 10 is 0.6739166666666667 where Timetaken is 156.98065359700558

## BEST PARAMETERS::

1.DEPTH=10, LEARNING RATE =0.1 , N\_ESTIMATORS=100

## 3. Confusion matrix:

For{ loss='deviance',n\_estimators=100,learning\_rate=0.1,  
max\_depth=10} on Validation set:  
Accuracy → 71.40%.

```
Confusion Matrix:
[[2008  319   41    0    4]
 [ 338 1616  549   40   39]
 [   27  419 1342  410   53]
 [    2   42  366 1576  343]
 [    4   28   66  341 2027]]
```

## 4. Classification Report:

Classification Report:				
	precision	recall	f1-score	support
Class-0	0.84	0.85	0.85	2372
Class-1	0.67	0.63	0.65	2582
Class-2	0.57	0.60	0.58	2251
Class-3	0.67	0.68	0.67	2329
Class-4	0.82	0.82	0.82	2466
accuracy			0.71	12000
macro avg	0.71	0.71	0.71	12000
weighted avg	0.71	0.71	0.71	12000

## Analysis:

1. Gradient Boosting on LDA components classifies the class 0 and class 4 better than the rest of the very efficiently with a precision of 0.84 and 0.82.
2. Highly misclassified class label is again class label 2.
3. Recall rate of class is 0 is highest followed by the class 4.
4. LDA performs less accurately than the PCA which can be seen when comparing Gradient boosting and SVM on precision and recall rate of all the classes.

# 9.PCA-KNN

## 1. hyperparameters:{n\_neighbors, weights, p}

**N\_neighbors:** (Default: 5) Number of neighbors to use for queries.

**Weights:** (uniform) In uniform, all the points in the neighborhood are weighted uniformly whereas in 'distance' points are weighted by the inverse of their distances.

**P:** (Default: 2) It is the power parameter for the metric. P=1 refers to the Manhattan distance, p=2 refers to the Euclidean distance,

## 2. Usage scenario/Design choice: (Number of parameters)

```
list=[2, 5, 10, 15, 20, 25, 35, 50, 75, 100]
```

KNN | Weights='distance' | p=2(Euclidean)

KNN | Weights='distance' | p=1 (manhattan)

KNN | Weights='uniform' | p=2 (Euclidean)

KNN | Weights='uniform' | p=1 (manhattan)

1. We have used **k=[2,5,10,15,20,25,35,50,75,100]** with the above parameters and cross validated.
2. Best accuracy was found at k=15, weights='distance', p=2 for which we validated our model on 12000 values (20% of training data).
3. An accuracy of 86.375% was achieved at the above hyperparameters.



4. **Design choice** for our problem of image classification with RF is

- PCA with 85% total explained variance.(42 components)
- KNeighborsClassifier(n\_neighbors=15,weights='distance',p=2).
- For Complete design, refer page 1.

### 3. Confusion matrix:

For{n\_neighbors=15,weights='distance',p=2}on Validation set:

Accuracy → 86.375%

```
Confusion Matrix:
[[2260  155   69    1   10]
 [ 101 2048  188   44   13]
 [   16  161 1865  204   37]
 [    1   54  193 1977  191]
 [    1    6   49  141 2215]]
```

### 4. Classification Report:

```
Classification Report:
              precision    recall  f1-score   support

   Class-0       0.95      0.91      0.93      2495
   Class-1       0.84      0.86      0.85      2394
   Class-2       0.79      0.82      0.80      2283
   Class-3       0.84      0.82      0.83      2416
   Class-4       0.90      0.92      0.91      2412

 accuracy              0.86      12000
 macro avg              0.86      0.86      0.86      12000
 weighted avg          0.86      0.86      0.86      12000
```

## Analysis:

1. KNN on PCA components classifies the class 0 and class 4 better than the rest of the very efficiently with a precision of 0.95 and 0.92.
2. Highly misclassified class label is again class label 2 and what closely follows is class label 3.
3. Recall rate of class is 0 is highest followed by the class 4.
4. Decision boundary of KNN is flexible, non-linear and reflects the classes very well. Thus, for Fashion MNSIT with 5 labels, it ends up giving a descent accuracy for the classes.

## 10.PCA-XGBOOST

### 1. hyperparameters:{max\_depth,objective,eval\_metric,num\_class}

**max\_depth (Default: 6):** Maximum depth of a tree to which it can extend the leaf node and it used to control the overfitting.

**Objective (Default= reg: linear)** It determines the loss function to be minimized. **Multi: softmax** is used to get the predicted class rather than the probabilities.

**Num\_class:** Defines the number of unique classes.

**Eval\_metric(Default: Error):** It is used for validation data. 'merror' is used for multi-class classification error rate.

## 2. Usage scenario/Design choice: (Number of parameters)

### RESULTS::--

Accuracy by XGBoost: 0.8813333333333333 Time taken 333.6746386119994 || Depth=5

Accuracy by XGBoost: 0.89025 Time taken 891.2257997380002 || Depth=6

Accuracy by XGBoost: 0.8899166666666667 Time taken 1351.3537010510008 || Depth=8

Accuracy by XGBoost: 0.8924166666666666 Time taken 1788.069563121 || Depth=10

Accuracy by XGBoost: 0.8934166666666666 Time taken 2272.261734103002 || Depth=20

Accuracy by XGBoost: 0.8918333333333334 Time taken 2685.0839237030013 || Depth=25

```
for depth in [5,6,8,10,20,25]:
    parameters_list = [{"objective", "multi:softmax"}, {"num_class", 5}, {"max_depth", depth}, {"eval_metric", "merror"}]
    n_rounds = 600
```

1. We have tested our model for the above hyperparameters with varying depth = [5,6,8,10,20,25] on 12000 values (20% of training data).
2. An accuracy of 89.34% was achieved at depth=20 keeping other parameters same as above.

### 3. Confusion matrix:

```
Confusion Matrix:
[[2264   84   32    0   10]
 [  86 2146  146   44   12]
 [  26  141 2002  158   41]
 [    1   41  135 2035  129]
 [    2   12   49  130 2274]]
```

### 4. Classification Report:

```
Classification Report:
              precision    recall  f1-score   support

   Class-0       0.95        0.95        0.95        2390
   Class-1       0.89        0.88        0.88        2434
   Class-2       0.85        0.85        0.85        2368
   Class-3       0.86        0.87        0.86        2341
   Class-4       0.92        0.92        0.92        2467

 accuracy              0.89              12000
  macro avg           0.89        0.89        0.89        12000
 weighted avg           0.89        0.89        0.89        12000
```

## Analysis:

1. XGBoost classifies all the classes very well as compared to the gradient boosting.

2. Class label 0 is classified with a precision of 0.95 and class label 4 with precision of 0.92.
3. XG boost will make the splits until the max\_depth is reached regardless of the loss being negative. It keeps the combined effect and ends up classifying better than the classical gradient boosting which can be seen from the results.

<b>CLASSIFIERS</b>	<b>MEAN ACCURACY (CROSS VAL)</b>	<b>ACCURACY (VALIDATION)</b>
<b>PCA-SVM</b>	<b>88.80</b>	<b>89.69</b>
<b>PCA-RF</b>	<b>87.38</b>	<b>87.40</b>
<b>PCA-DT</b>	<b>81.20</b>	<b>81.52</b>
<b>LDA-SVM</b>	<b>72.87</b>	<b>72.76</b>
<b>PCA-NB</b>	<b>62.08</b>	<b>62.08</b>
<b>PCA-LR</b>	<b>69.47</b>	<b>69.475</b>
<b>PCA-GB</b>	<b>88.11</b>	<b>88.275</b>
<b>LDA-GB</b>	<b>71.40</b>	<b>71.40</b>
<b>PCA-KNN</b>	<b>86.62</b>	<b>86.375</b>
<b>PCA-XGB</b>	<b>89.34</b>	<b>89.34</b>

## **Result of above comparisons→**

1. It Can be seen from the above analysis that class 0 is least misclassified by all the algorithms.
2. Class label 2 is most misclassified by all the algorithms.

3. Naïve Bayes tends to be the worst algorithm for our dataset as it is a linear classifier and makes the strong data distribution assumption that any two features are independent for a class label. It calculates the conditional probability of all the classes assuming each feature as independent.
4. Best algorithm with least number of misclassifications is the SVM applied on PCA which classifies extremely well for our dataset. Also, SVM performs well when the sparsity is high which somehow holds true for our dataset.
5. XG Boost also provided the promising results for the classification problem as it provided the improved precision on gradient boosting given the fact that it reduces the overfitting and continues with the splitting until the max depth is reached(regardless of loss being negative).

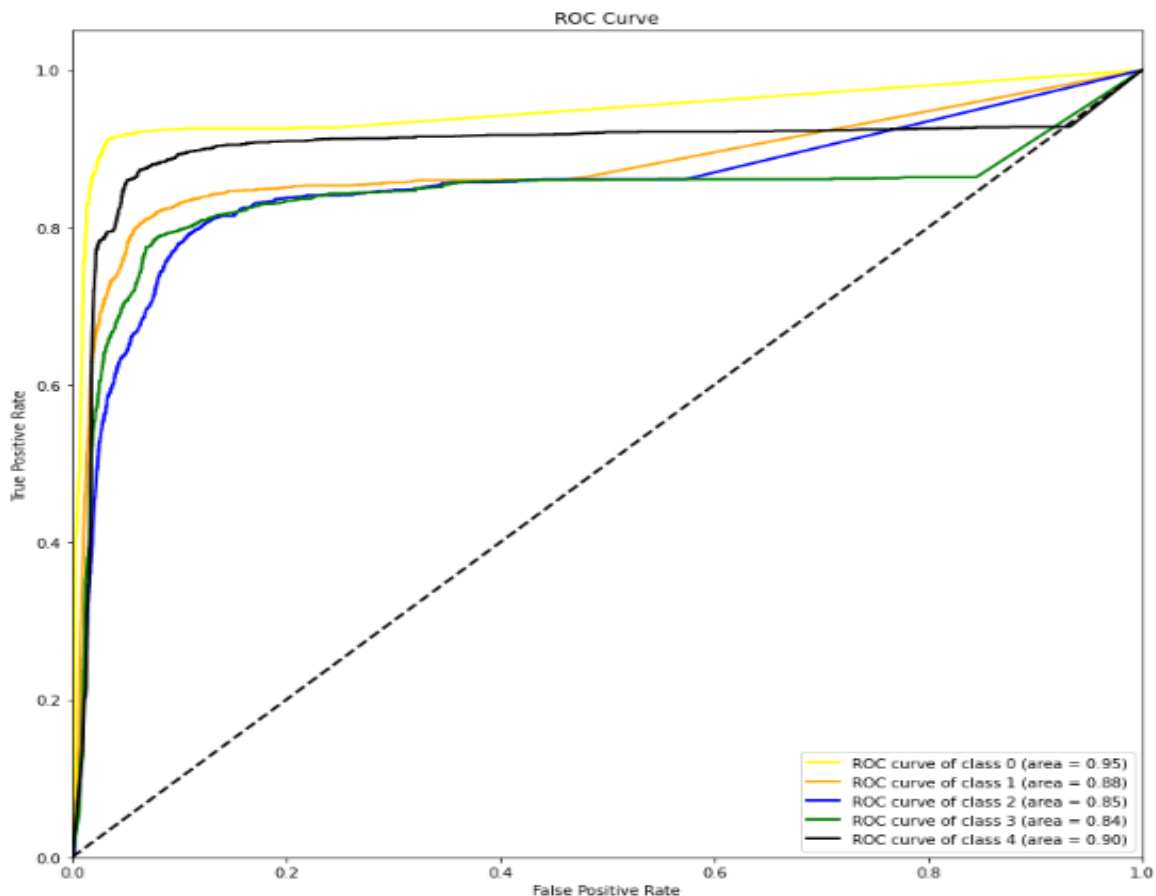
# ROC

## NOTE:

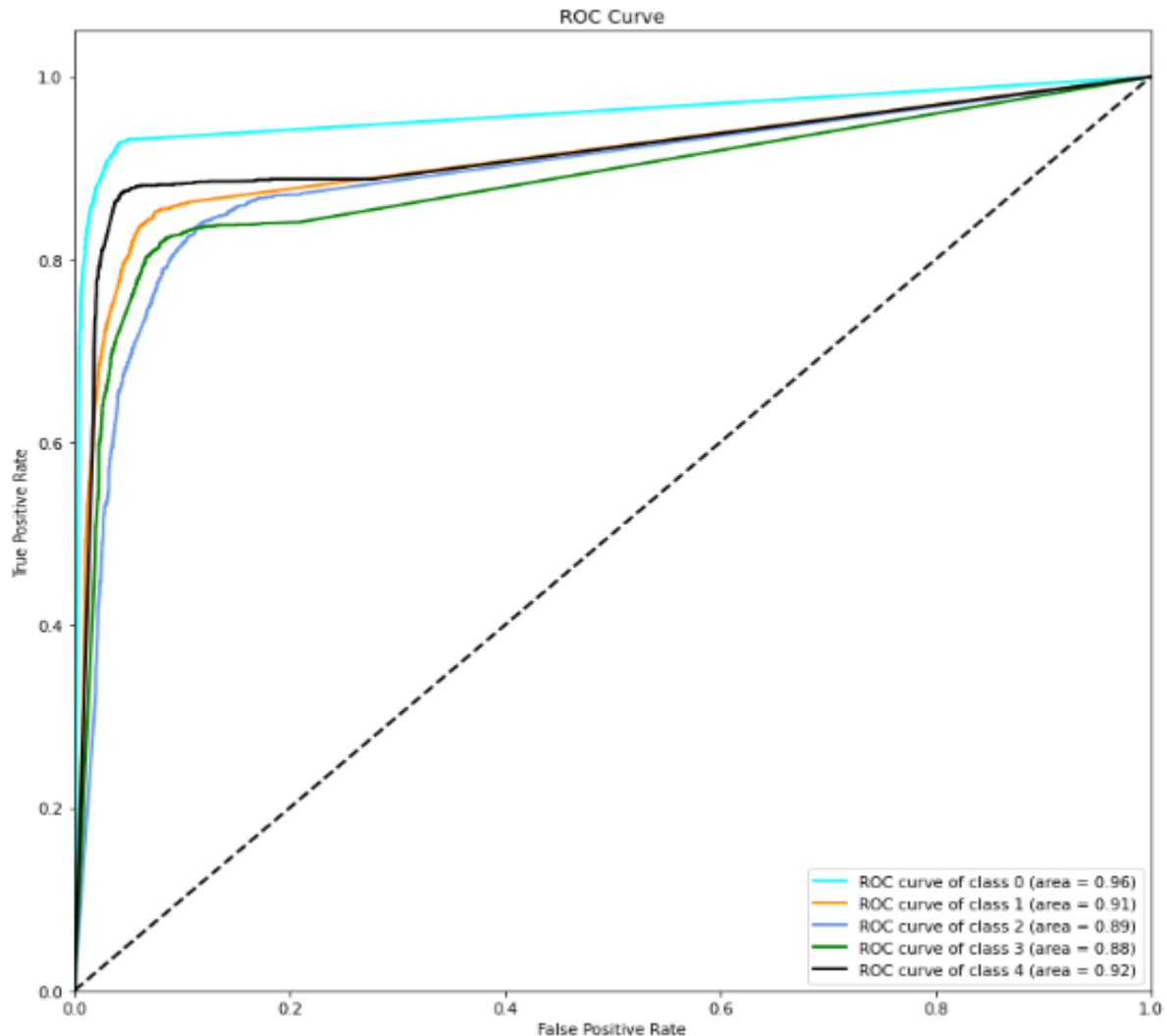
1. We have made ROC curves for all the 10 methods we tried except 'XGBoost'.
2. We have also made ROC curves to determine the effect of parameters on the accuracy of some models.
3. We will be analyzing few ROC curves here.

## 1. DECISION TREE -----

- a. **Hyperparameters:** Max-depth=15, criterion='gini' (2<sup>nd</sup> Best chosen by cross Val score.)



**b. Hyperparameters:** Max-depth=15, criterion='entropy'  
(Best chosen by cross Val score.)



## ANALYSIS:

1. For the hyperparameter criterion='gini', validation accuracy was slightly less than the one we got using criterion='entropy'.
2. The above results can be confirmed by analyzing the above ROC curves.



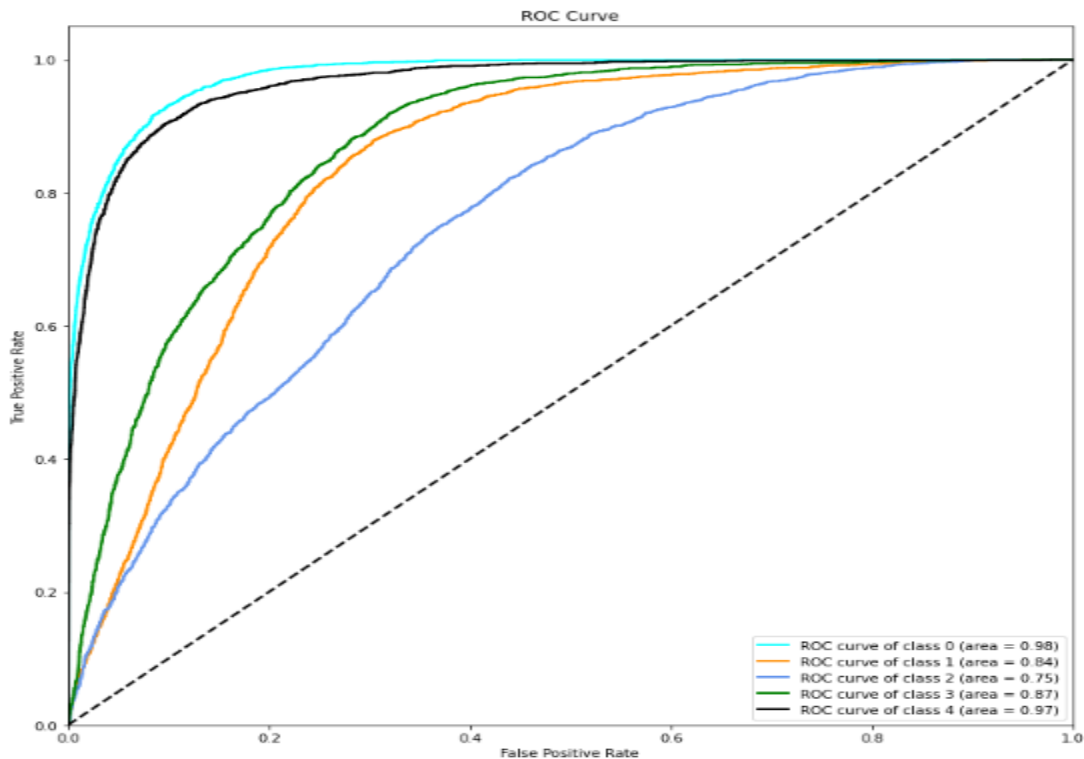
3. For all the classes, the classifiers made when using criterion='entropy', the AUC is better than the one we got in 'gini'.

Class	Gini	Entropy
0	0.95	0.96
1	0.88	0.91
2	0.85	0.89
3	0.84	0.88
4	0.90	0.92

4. Thus, both the criterion= {'gini', 'entropy'} classifies class label 0 very well followed by the class label 4 but the criterion='Entropy' outperformed the 'gini' by all means in all the classes which can be seen. Same result we concluded using Confusion matrix, classification report and other graphs.

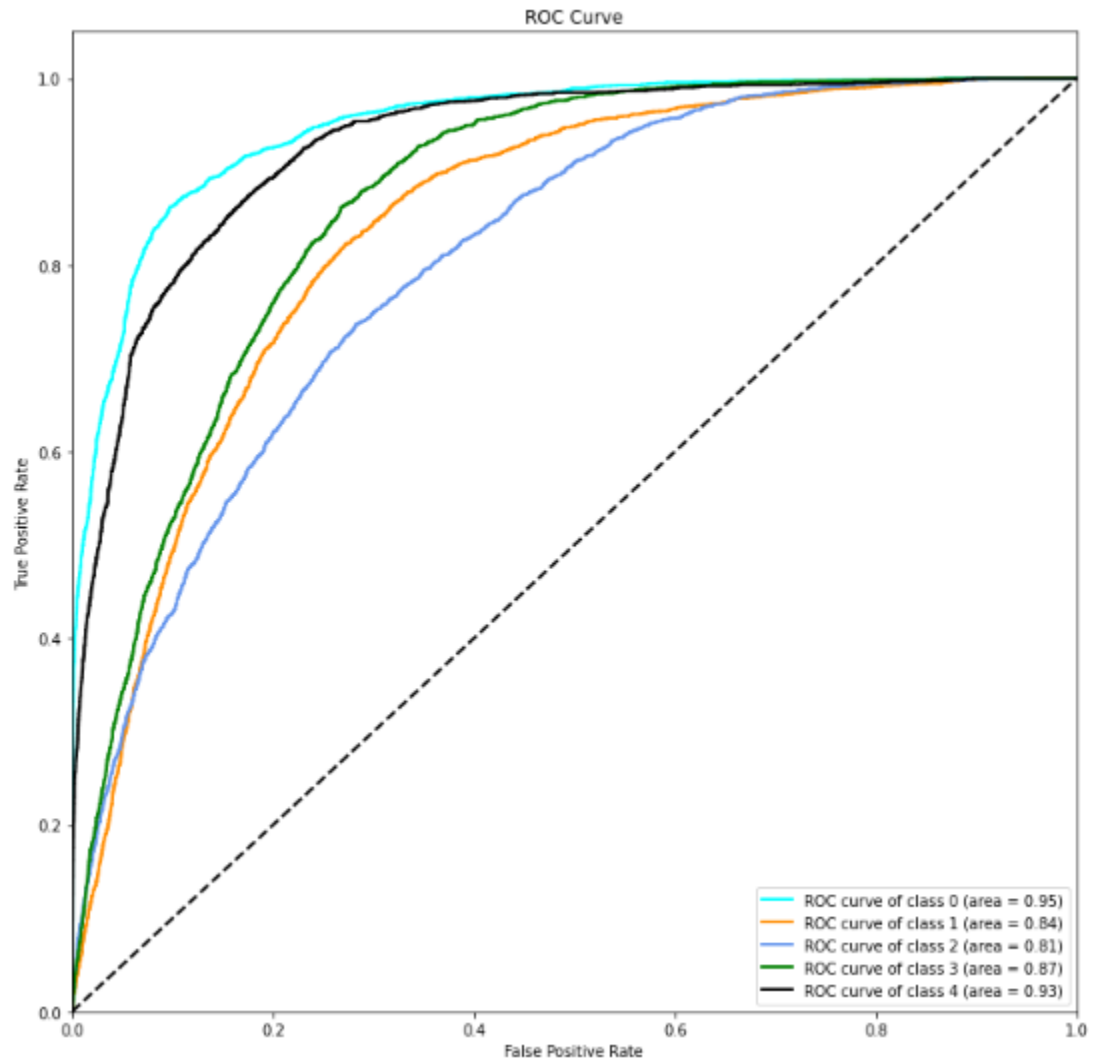
## 2. Logistic Regression:

**Hyperparameters:** Solver=lbfgs, max\_iter=4000



### 3. Naïve Bayes

**Hyperparameters:** priors=None, var\_smoothing=1e-09



#### ANALYSIS:

1. Both the linear classifiers ' Logistic Regression' and 'Naïve Bayes' perform worst as compared to the other classifiers which can be clearly seen from the above ROC plots as well.
2. Logistic regression outperforms the Naïve Bayes somehow which can be seen here:

Class	Naïve Bayes	Logistic Regression
0	0.95	0.98
1	0.84	0.84
2	0.81	0.75
3	0.87	0.87
4	0.93	0.97

3. It can be seen that the class label 0 and class label 0 is classified well by looking at the Auc by the Logistic regression which is due to the usage of lbfgs which is used for the multi class problems.
4. Naïve Bayes's bad performance is the result of the strong data distribution assumption made by the classifier i.e. the features are independent of the class label.

**NOTE: Remaining ROC's for all the methods can be found in the ipynb notebooks.**

# Benefit of using some data as test data from the train data.

1. In supervised learning, the model has to be trained with some data having labeled input with known outputs. Once the training of the model is done, the model has to be tested on the **test data which is not exposed earlier to the model during training**.
2. To achieve this, we need to split the dataset into train and test sets using: `sklearn.model_selection.train_test_split` giving required parameters.
3. **Problem and Need:** If by any case, the test data is exposed during the training of the model, it will result in **over fitting** and overestimating the accuracy of the model when the same training data is tested. This model will not generalize well on the new test data. As a result, the model will become potentially unusable to deploy for practical applications. This problem is also known as **Data Leakage**.
4. **Solution:** To analyze the generalization of the model we need to test the model with novel and unseen data. Thus, there comes a need of the test dataset.
  1. In order to solve the problem of data leakage, we can divide the dataset into train-validate-test sets.
  2. Thus, the validation dataset which is taken from the training set can be used as the initial test set before performing the actual testing on test dataset.
  3. Secondly, the k-fold cross validation can also be used to properly split the datasets with the cross validation split strategy.

**Overall, we split the dataset into train and test sets to prevent over-fitting, data leakage and isolate the test dataset so that it is not exposed to model while the model is learning.**

## Example:

**# splitting the dataset into train and test sets in 80:20.**

```
X_train,X_test,y_train,y_test=train_test_split(train.data,train.target,test_size=0.20,train_size=0.80,random_state=42)
```

**# further splitting the train set into train and validate set.**

```
X_train1,X_val,y_train1,y_val=train_test_split(X_train,y_train,test_size=0.25,random_state=42)
```

**# for k-fold cross validation. Here, CV=10 does 10-fold splitting performing test on 1 fold in each split.**

```
cross_val_score(estimator, X_train, y_train, CV=10)
```

1. By using Cross Val score, we were able to find the best parameters keeping the basis as accuracy.
2. For the best accuracy given for the best-found parameters at the previous step, we further validate our model by using 20% of the training data(60000) as the validation data.
3. By following this we were able to evaluate our model on various parameters for each classical method.

**CROSS VAL COMAPARISON (VALIDATION COMPARISON FOR BEST PARAMETERS FOUND USING CROSS VAL IS ALREADY DONE IN PREVIOUS SECTIONS.)**

## **PCA-SVM**

**Hyperparameters: [C=n, kernel={'rbf', 'poly'}]  
C= [0.1,0.5,1,2,3,5,10,11,12,15,30]**

### **Cross Val:**

**Mean Accuracy: 0.8314583333333333 at c= 0.1 Time taken :: 409.22398139099823**

**Mean Accuracy: 0.8612500000000001 at c= 0.5 Time taken :: 312.48221586699947**

**Mean Accuracy: 0.8703125 at c= 1 Time taken :: 275.2137646659976**

**Mean Accuracy: 0.8778333333333335 at c= 2 Time taken :: 257.40431245200307**

**Mean Accuracy: 0.8806041666666667 at c= 3 Time taken :: 247.66984448600124**

**Mean Accuracy: 0.88425 at c= 5 Time taken :: 244.5811389540031**

**Mean Accuracy: 0.887875 at c= 10 Time taken :: 241.1030371789966**

**Mean Accuracy: 0.8877708333333334 at c= 11 Time taken :: 242.08307200200215**

**Mean Accuracy: 0.8880833333333333 at c= 12 Time taken :: 240.36959404299705**

**Mean Accuracy: 0.8888124999999999 at c= 15 Time taken :: 243.86396649600647**

**Mean Accuracy: 0.8888541666666667 at c= 20 Time taken :: 245.7119200629968**

**Mean Accuracy: 0.8884166666666667 at c= 30 Time taken :: 260.56993236299604**

### **BEST CHOSEN:**

**[C=11.7/12, kernel='rbf']**

### **VALIDATION(20%)**

- 1. [C=12, kernel='rbf'] → 89.70% accuracy**
- 2. [C=10, kernel='rbf'] → 89.64% accuracy**
- 3. [C=11.7, kernel='rbf'] → 89.69% accuracy**
- 4. [C=10, kernel='poly'] → 87.64% accuracy**

**Thus, doing this we were able to get the best parameters and cross checked for the same.**

# RANDOM FOREST

## Hyperparameters:

`[depth=[5,10,15,None],n_estimators=[5,10,50,100,150],criterion={'gini','entropy'} ]`

## Cross Val Score:

### CHOOSING THE BEST PARAMETERS. ¶

Here we are choosing max\_depth=None and Number of trees=150 because these parameters are giving an accuracy of approximately higher than the accuracy given by the parameters max\_depth=None and Number of trees=150.

Computational time for both the set of hyperparameters is given below:

1. Mean Accuracy: 0.8700833333333332 at depth= None and when no of trees= 150 Time taken :: 568.7620693479985

2. Mean Accuracy: 0.8692291666666666 at depth= None and when no of trees= 100 Time taken :: 378.6132203479974

### RESULT::----

1. NO OF TREES= 150 | DEPTH= NONE | CRITERION='gini' → 87.0

2. NO OF TREES= 150 | DEPTH= NONE | CRITERION='entropy' → 87.38

3. Entropy is better than gini in our case.

4. No of trees=150 and depth=none is giving better accuracy in both the cases. Different combination of depths and no of trees are tried.



## **BEST CHOSEN:**

**Hyperparameters: {depth=None, n\_estimators=150,  
criterion='entropy'}**

### **On Validation:**

**Time: 152.843**  
**Accuracy: 87.40%**

## **DECISION TREE**

### **Hyperparameters:**

**[depth=[3,5,7,10,15,20,50,75,None], criterion={'gini', 'entropy'}  
]**

### **Cross Val Score:**

#### **Criterion='Gini'**

**Mean Accuracy: 0.4851458333333333 at depth= 3 Time taken: 8.955950443996699**

**Mean Accuracy: 0.6266041666666666 at depth= 5 Time taken: 14.156486516993027**

**Mean Accuracy: 0.7079166666666667 at depth= 7 Time taken: 19.129805596996448**

**Mean Accuracy: 0.7827083333333333 at depth= 10 Time taken: 26.32768580699485**

**Mean Accuracy: 0.80975 at depth= 15 Time taken: 35.28414714799874**

**Mean Accuracy: 0.8049583333333332 at depth= 20 Time taken: 40.04835457800072**

**Mean Accuracy: 0.7981874999999999 at depth= 50 Time taken: 44.41565103500034 -**

**Mean Accuracy: 0.7981874999999999 at depth= 75 Time taken: 45.57565351700032 -**

**Mean Accuracy: 0.7981874999999999 at depth= None Time taken: 46.3597181750010:  
same**

### **Criterion='entropy'**

Mean Accuracy: 0.4799166666666667 at depth= 3 Time taken: 24.302092085999902

Mean Accuracy: 0.6734791666666667 at depth= 5 Time taken: 39.1497254829992

Mean Accuracy: 0.748125 at depth= 7 Time taken: 52.45439914400049

Mean Accuracy: 0.8011458333333333 at depth= 10 Time taken: 67.9292816729976

Mean Accuracy: 0.8120625 at depth= 15 Time taken: 79.98422932899848

Mean Accuracy: 0.8080833333333333 at depth= 20 Time taken: 81.98011783200127

Mean Accuracy: 0.8074791666666666 at depth= 50 Time taken: 83.29971088900129

Mean Accuracy: 0.8074791666666666 at depth= 75 Time taken: 82.12922389600135

Mean Accuracy: 0.8074791666666666 at depth= None Time taken: 84.96015837600135

### **RESULT::**

BEST DEPTH ⇒ 15 IN ALL PARAMETER TUNING. → {'entropy','gini'}

BEST CRITERION ⇒ Entropy

Time complexity of 'entropy' is more than double to that of 'gini'.

### **BEST CHOSEN:**

**Hyperparameters:{depth=15, criterion='entropy'}**

### **On Validation:**

**Time: 8.8**  
**Accuracy: 81.52%**

# **GRADIENT BOOSTING**

## **1. Hyperparameters:**

```
{  
  max_depth=[5,10,15,20,50]  
  n_estimators=[5,10,25,50,100],  
  learning_rate=[0.001,0.01,0.05,0.1,1]  
  loss='deviance'}
```

## **2. CROSS VAL SCORE:**

**A snippet of just few results:**

**RESULTS ::-**

Mean Accuracy at learning rate=0.01 , n\_estimators= 10 and at max\_depth=10 is 82.34% where Timetaken is 752.5

Mean Accuracy at learning rate=0.01 , n\_estimators= 10 and at max\_depth=20 is 80.36% where Timetaken is 1186.94

Mean Accuracy at learning rate=0.01 , n\_estimators= 10 and at max\_depth=30 is 79.45% where Timetaken is 1366.96

Mean Accuracy at learning rate=0.01 , n\_estimators= 50 and at max\_depth=10 is 84.33% where Timetaken is 3498.92

Mean Accuracy at learning rate=0.01 , n\_estimators= 50 and at max\_depth=20 is 82.50% where Timetaken is 6123.34

Mean Accuracy at learning rate=0.01 , n\_estimators= 50 and at max\_depth=30 is 80.56% where Timetaken is 6983.46

-- NOTE -- Eliminating depth=20,30 because it is increasing the time complexity and resulting in less accuracy than depth=10 .[average change in accuracy is 2% ]

-- NOTE -- Eliminating number of estimators=10,50 because keeping depth and learning rate constant gives an exponential increase in the accuracy with changing estimators. SO, KEEPING THE DEFAULT VALUE OF N\_ESTIMATORS i.e 100 for further analysis.

Mean Accuracy at learning rate=0.1 , n\_estimators= 100 and at max\_depth=10 is 88.11% where Timetaken is 5895.38

Mean Accuracy at learning rate=1 , n\_estimators= 100 and at max\_depth=10 is 85.9% where Timetaken is 4317.33

## **RESULT→**

**BEST parameters using cross val score are:**

→learning rate=0.1

→n\_estimators= 100

→max\_depth=10

## **BEST CHOSEN**

**Hyperparameters:**

**{max\_depth=10, n\_estimators=100, lr=0.1, loss='deviance', criterion='gini'} |**

**On validation:**

**Time: 1422**  
**Accuracy: 88.275%**

# KNN

## 1. Hyperparameters:

{k=[2,5,10,15,20,25,35,50,75,100], weights={'distance',  
'uniform'},  
p=[1,2]}

## 2. CROSS VAL:

KNN | Weights='distance' | p=1 (manhattan)

Mean accuracy at k= 2 is 0.8179833333333333 Time Taken 78.98695889599912

Mean accuracy at k= 5 is 0.8484666666666667 Time Taken 93.69966886699694

Mean accuracy at k= 10 is 0.8549833333333334 Time Taken 103.48188868600118

Mean accuracy at k= 15 is 0.85685 Time Taken 110.28686005000054

Mean accuracy at k= 20 is 0.8566833333333334 Time Taken 115.64875962799852

Mean accuracy at k= 25 is 0.8544833333333333 Time Taken 120.79132175000268

Mean accuracy at k= 35 is 0.8524333333333333 Time Taken 127.48865861000377

Mean accuracy at k= 50 is 0.8480166666666666 Time Taken 136.4493232770037

Mean accuracy at k= 75 is 0.8422000000000001 Time Taken 144.30264079300105

Mean accuracy at k= 100 is 0.8374166666666666 Time Taken 156.45156359400426

Similarly, for the other combinations, results were analyzed and the following hyperparameters were found to work better than the rest. For full results please consult markdown in notebook.

## **BEST CHOSEN:**

**Hyperparameters:**

**{k=15, weights='distance', p=2}**

**On validation:**

**Time: 9.94**

**Accuracy: 86.375%**