Course Project: [CSE342] Statistical Machine Learning

Arnav Goel
2021519
B. Tech, Computer Science with Artificial Intelligence
Indraprastha Institute of Information Technology
arnav21519@iiitd.ac.in

Mehar Khurana
2021541
B. Tech, Computer Science with Artificial Intelligence
Indraprastha Institute of Information Technology
mehar21541@iiitd.ac.in

***Abstract*—The paper presents a summary of the various statistical and machine learning techniques used for performing multi-class classification on a given data set. The given data set consists of 20 classes corresponding to 10 different fruits and whether they are ripe or raw.**
To perform this classification, we start with data visualisation followed by data pre-processing. Data pre-processing involves outlier detection techniques (such as Local Outlier Factor and Isolation Forest) and data scaling techniques (such as Standard Scalar and Quantile Transformer). We follow this with dimensionality reduction algorithms such as Principal Component Analysis (PCA) and Linear Discriminant Analysis (LDA) and clustering techniques to add cluster IDs as additional features to our data set. We then experiment with various machine learning algorithms like Logistic Regression, K-Nearest Neighbors, Decision Trees and Random forest classifiers. In the end we try improving our model accuracy by trying ensemble techniques like Gradient Boosting, ADA Boost, Bagging Classifiers, Voting Classifiers, etc and deep learning techniques like Convolution Neural Networks (CNNs).
We evaluat the performance of the models using cross-validation and compared their accuracy and computational efficiency. Our results show that logistic regression with a 'newton-cg' solver outperformed other models with an accuracy of 85 on the public data. The Convolution Neural Network gives the highest accuracy of 84 on the private data.
Overall, this project demonstrates the effectiveness of statistical machine-learning techniques in multi-class classification tasks and highlights the importance of outlier detection and dimensionality reduction in improving the accuracy of the Machine Learning models.

***Index Terms*—component, formatting, style, styling, insert**

I. Introduction

The paper presents a summary of the various Machine Learning and Statistical techniques which were implemented and applied on the data set provided to us as part of the course project for CSE342. The training data set is given labelled to us with 20 classes and comprised of 1216 - 64 x 64 images stored in a matrix of size 1216 x 4096 with each column corresponding to a pixel in the image. The model was then tested against unlabelled testing data comprising of 415 images which are to be classified amongst the 20 classes mentioned.

We start by pre-processing the data using outlier detection techniques, followed by data scaling, normalisation and feature extraction techniques (such as clustering and dimensionality reduction) as mentioned above. For testing and fine-tuning the hyper-parameters of the various data pre-processing techniques, we use a baseline Logistic Regression Model (i.e. tuned with **max-iters = 1000** and **solver = newton-cg**). We use the cross-validation accuracies generated by various hyper-parameters as a comparison metric to gauge which hyper-parameter will give the best possible model in the end.

II. Data Preprocessing

A. Data Visualisation

We attempt to visualise the training data provided to us in order to help us understand the data better. First, we see how many members are there in each class to get an understanding of how skewed the data is. We find that the training data is slightly skewed with the class "$Leeche-Raw$" occurring only 38 times in the training data whereas the class "$Banana-Ripe$" occurs 86 times.

We plot the data in the form of a histogram and note that the data is Non-Gaussian in nature. Thus, we need to detect outliers and scale this data to improve accuracy under the various classification models we are going to try.

B. Outlier Detection

We start by performing anomaly detection on the entire training data. We use two techniques provided by the sklearn library i.e. Local Outlier Factor (LoF) and Isolation Forest Ensemble.

1) ***Local Outlier Factor (LoF):*** The Local Outlier Factor (LoF) method in the sklearn library implements the LoF technique which tries to use the Local Reachability Density of a point from its k-nearest neighbors for detecting outliers. Thus the main hyper-parameters to be tuned while applying LoF is **n-neighbors** or the value of k and **contamination** which indicates the percentage of contamination we expect in the entire data-set. We do not tweak the contamination hyper-parameter as we do not know the amount of outliers in our data. We let the method decide the threshold on its own instead of providing a contamination parameter. We tweak the n-neighbors hyper-parameter and the results for the cross-validation accuracy are summarised in Table 1. Table 1 shows that the highest accuracies are achieved with n-neighbors = 9 and 11 and we use these 2 metrics for our outlier detection as they both remove 6 outliers.

2) ***Isolation Forest:*** The Isolation Forest ensemble consists of several isolation trees, where each tree is trained on a random subset of the data. Each tree in the ensemble is trained to isolate the anomalies by recursively partitioning the data

TABLE I: LoF Results

n-neighbors	n-outliers	accuracy
6	8	79.622
7	8	79.677
8	5	79.672
9	6	**79.887**
10	4	79.588
11	6	**79.876**
12	3	79.566
13	3	79.544
14	4	79.532

into smaller subsets.
Running an iterative search on this anomaly detection algorithm with baseline linear regression yields the best results, again, at 6 outliers.

C. Clustering

After removing the outliers, we try to perform clustering on the training data. The idea is to use the Cluster IDs given to each data point by the clustering algorithm as an additional feature to our training data. We try clustering by the four algorithms given below:

*1) **k-Means Clustering**:* k-Means clustering needs you to specify the number of clusters you want from as a hyper-parameter to the KMeans algorithm from the sklearn library. It repeats this process till the centroids converge. We run the clustering algorithm from $k = 2$ to $k = 60$ and use Silhouette Score Analysis from the sklearn library to try and find the best k-value for clustering the data. Upon running the clustering algorithm and finding the silhouette scores for each $k - value$, we observe that the highest silhouette score value came out to be only **0.1** at **k=7**. Silhouette score analysis says that a score close to 0 indicates overlapping clusters whereas a score close to 1 indicates close-to-perfect clustering. This low score indicates k-means is not a good clustering metric for the given data-set and we must try some other clustering.

*2) **Fuzzy c-means Clustering**:* Fuzzy c-means clustering like k-means requires us to set the number of clusters we are expecting to form. We try the same range as k-means for fuzzy c-means as well and try to judge the clustering done by using silhouette score analysis. We find that the silhouette scores are more or less same as K-Means clustering and thus we try to find a clustering algorithm where we do not specify clusters beforehand.

*3) **Mean Shift Clustering**:* The first algorithm we try is Mean Shift Clustering which tries to use a bandwidth hyper-parameter to find the appropriate clustering for the dataset. We try two techniques, first the $estimate - bandwidth$ method in the sklearn library to estimate the bandwidth which gave the bandwidth of 16.75. We ran mean-shift clustering on this bandwidth. This gave a silhouette score of -0.04. We then try estimating the bandwidth by reducing it further only to find our silhouette score to taper at 0.02 which was not any better than the previous two algorithms we used.

*4) **DBSCAN**:* We run the DBSCAN clustering algorithm from the sklearn. It however clusters all the data points into one cluster and labels them as noise. Thus, this clustering algorithm is not ideal too.

Hence, we drop the idea of using cluster IDs as additional features for the data. This is happening probably because of how the data is distributed and maybe that it exists in form of a ring which is preventing an ideal clustering from occurring.

D. Scaling / Transforming the Data

Since it is recommended that the data inputted to linear models be Gaussian in nature, we try to scale the data and convert it into a Gaussian distribution. We use the **Standard-Scalar()** and the **Quantile Transformer()** methods from the sklearn library. However, upon performing this transformation, the accuracy of the baseline Logistic Regression model drops rapidly to about 75% and thus we decide to drop the idea to scale or transform the data by the given methods in our data pre-processing.

E. Dimensionality Reduction

We perform dimensionality reduction to extract features and better help in the classification downstream task. We use two dimensionality reduction techniques i.e. Principal Component Analysis which tries to maximise the variance explained in the principal components and Linear Discriminant Analysis which maximises the separation between the two classes.

*1) **Principal Component Analysis (PCA)**:* We use the PCA function from the $sklearn.decomposition$ package which we use to find the eigenvalues and eigenvectors of the covariance matrix of the training data. We then plot the graph as shown in Figure 1

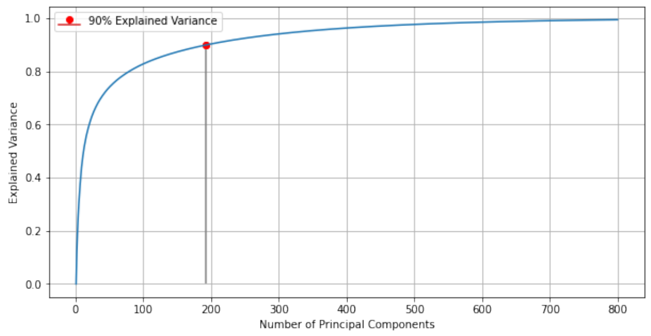


Fig. 1: Explained Variance v/s No. of Principal Components

The plot in Figure 1 shows that the we get 90% variance explained by about 200 Principal Components alone. This reduces the dimensions of our data from 4096 to 200 which is a 95% decrease in our dimensions not a very big loss in information. In fact it reaches around 95% variance by 300 components and tapers to 100% variance around 650 components. This speeds up the computation of our data and helps improve accuracy.

2) ***Linear Discriminant Analysis (LDA):*** We use the $LinearDiscriminantAnalysis()$ function in the $sklearn.discriminant\_analysis$ after performing Principal Component Analysis. This helps in separating the data belonging to the 20 classes when we reduce the data further to 19 features after performing LDA. Since LDA reduces the number of features of your data to (number of classes - 1), we input n_components as 19 and perform LDA on the PCA undergone data.

Fine-Tuning PCA and LDA:
Table II summarises the results after performing cross-validation using various different combinations of Principal Components with the baseline Logistic Regression model. The first two columns show the number of components we have taken with PCA and LDA respectively. We use the Grid Search method in the sklearn library to find the best Principal Component parameters. We can see that reducing LDA components below 19 reduces accuracy significantly and hence must be kept at 10 always. As seen in the graph that 300 principal components explain around 95% variance, we run this Grid Search only for components greater than 300.

PCA	LDA	accuracy(LR)
300	14	75.676
300	19	80.532
320	19	**80.584**
340	19	80.292
360	19	**80.634**
380	19	80.072
400	19	79.303
420	19	79.544

TABLE II: Fine-Tuning PCA and LDA

We can see that the range of 330-370 is an ideal range of principal components to choose from. We choose an arbitrary number in the middle which was giving the highest accuracy on multiple iterations i.e. 356 principal components which roughly translates to 98% explained variance.

III. Training - Machine Learning Techniques

In this section, we discuss the various Machine Learning Models we use to create a classifier. We perform a Grid Search for each of these models (with cross validation) to find the best hyper-parameters for the model. We then list out the final accuracies attained by each model in the end.

A. *Logistic Regression*

We use the **LogisticRegression()** method from the sklearn-library for creating this classifier. We fix the maximum iterations to 1000 as the model converges by then. We find that the highest accuracy of on the validation set is achieved by using a **newton-cg** solver. Also since the number of data points in each class in not the same, we fix the class_weight hyper-parameter as **balanced**.

B. *Decision Tree Classifier*

We use the **DecisionTreeClassifier()** method from the sklearn library for creating this classifier. We get the following hyper-parameters after performing Grid Search:

- max_depth: refers to the maximum depth upto which we can make the decision tree. **Set to 90**
- min_samples_leaf: refers to the number of samples needed to stop splitting and called it a leaf node. **Set to 3**
- min_samples_split: refers to the minimum number of samples needed to split an internal node. **Set to 6**

C. *k-Nearest Neighbors*

We use the **kNeighborsClassifier()** method from the sklearn library for this. We test this for $k = 2$ to $k = 15$ to find the best value of k for performing the classification. We generally get the highest accuracies for k values from 2 to 5.

D. *Naives Bayes Classifier*

We use the **GaussianNB()** method of the sklearn library to create this classifier. We do not tune any hyper-parameters in this model and it gives a very decent accuracy without any tuning.

Average Validation Accuracies:

Classifier	Accuracy
Logistic Regression	**83.4**
Naive Bayes Classifier	83.15
kNN Classifier	82.75
Decision Tree Classifier	62.5

TABLE III: Validation Accuracies

IV. Training - Ensemble Methods

There is a upper threshold on the maximum accuracy one can achieve by using simple and normal machine learning techniques. We hence use Ensemble Techniques which use Boosting or Bagging to help increase accuracy of our predictions. We list the various ensemble methods we tried to boost our final accuracy:

A. *Random Forest Classifier*

The Random Forest Classifier uses various Decision Trees and performs Feature Bagging and Voting on their predictions to boost the accuracy of the Decision Tree Classifier. We get the following hyper-parameters after performing Grid Search on the **RandomForestClassifier()** method from the sklearn library:

- n_estimators: refers to the number of DT estimators we want to create for this classifier. **Set to 1000**
- max_depth: refers to the maximum depth upto which we can make the decision tree. **Set to 90**
- min_samples_leaf: refers to the number of samples needed to stop splitting and called it a leaf node. **Set to 3**
- min_samples_split: refers to the minimum number of samples needed to split an internal node. **Set to 6**

B. *Gradient Boosting Classifier*

The Gradient Boost Classifier also boosts Decision Trees to perform classification on the given data. We get the following hyper-parameters after performing Grid Search on the **GradientBoostingClassifier()** method from the sklearn library:

- n_estimators: refers to the number of DT estimators we want to create for this classifier. **Set to 400**
- max_depth: refers to the maximum depth upto which we can make the decision tree. **Set to 90**
- min_samples_leaf: refers to the number of samples needed to stop splitting and called it a leaf node. **Set to 3**
- min_samples_split: refers to the minimum number of samples needed to split an internal node. **Set to 6**

C. *ADABoost Classifier*

The ADABoost Classifier uses a weak classifier and adjusts weights of incorrect instances to better or correct the predictions and thus the overall model. We get the following hyper-parameters after performing Grid Search on the **ADABoostClassifier()** method from the sklearn library:

- estimator: refers to our base estimator which we want to boost. We use the **Logistic Regression** model here.
- n_estimators: refers to the number of estimators we want to create for this classifier. **Set to 400**
- learning_rate: **Set to 0.01**

D. *Bagging Classifier*

The Bagging Classifier is a type of ensemble model that combines multiple independent classifiers trained on different subsets of the training data to create a strong model. We get the following hyper-parameters after performing Grid Search on the **BaggingClassifier()** method from the sklearn library:

- estimator: refers to our base estimator which we want to boost. We use the **Logistic Regression** model here.
- n_estimators: refers to the number of estimators we want to create for this classifier. **Set to 100**

E. *Stacking Classifier*

The Stacking Classifier is a type of ensemble model that combines multiple heterogeneous classifiers by training a meta-classifier on their predictions. For the base classifier, we use Logistic Regression, Naive Bayes, and the Bagging Classifier. We then use the Logistic Regression model we trained as our final estimator for this classifier.

F. *Voting Classifier*

The Voting Classifier is a type of ensemble model that combines multiple classifiers by taking their individual predictions and aggregating them into a final prediction. We use 4 different classifier models which we trained for training the voting classifier. They are listed as follows:

- Logistic Regression
- Naive Bayes Classifier
- Bagging Classifier
- Stacking Classifier

We additionally use **hard** voting as the hyper-parameter which selects the class with the maximum occurrence in the predictions as the final prediction of the voting classifier.

Average Validation Accuracies:

TABLE IV: Validation Accuracies

Classifier	Accuracy
Random Forest Classifier	80.5
Gradient Boosting Classifier	71.5
ADABoost Classifier	82.17
Bagging Classifier	83.4
Stacking Classifier	83.15
Voting Classifier	**83.82**

V. Training - Deep Learning Techniques

A. *Neural Networks (MLPClassifier)*

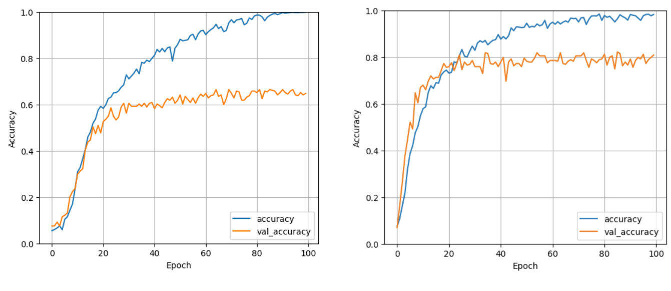
We implement a basic Neural Network using the $MLPClassifier$ model from the $sklearn$ library, with the $adam$ solver.
Running a grid search along with 5-fold cross validation on this model for the optimal hidden layer sizes yielded two optimal architectures: one with a depth of 3 and $hidden\_layers$ = (300, 60) and another with depth of 5 and $hidden\_layers$ = (448, 119, 170, 116). These gave maximum validation accuracies of 79% and 81% respectively.
We witnessed substantial drops in validation accuracies on using PCA and LDA as the feature selection algorithms along with this model, and thus, decided against using them before the MLPClassifier. The data were standardized using the $StandardScaler$ from $sklearn$.

TABLE V: Validation Accuracies with MLPClassifier

Dim. Reduction	Accuracy
Without PCA or LDA	76.02
With PCA	76.12
With LDA	35.8
With Both	**76.74**

B. *Convolutional Neural Networks (tensorflow)*

As the validation accuracies we obtained using the $MLPClassifier$ are fairly high, and since the data set is comprised of 64 x 64 images, we train and evaluate CNNs for this classification problem.
We use the tensorflow library to create a CNN. Initially, we use 3 convolution layers with 64, 32, and 32 channels (in order of increasing depth), each followed by a MaxPooling layer of size (2, 2). ReLU is used as the activation function in all 3 convolution layers.
We also add a flatten layer and two dense layers of size 64 and 20 respectively. The first dense layer uses ReLU for activation and the second uses the sigmoid function, since this is a classification problem. The 20 nodes in the final layer output the activation values for each class, and we label the samples based on the max of these values.



(a) CNN without regularization (b) CNN with regularization

Fig. 2: Training (blue) and Validation (orange) accuracies

For backpropagation, we calculate the loss using SparseCategoricalCrossEntropy loss.

The above architecture is observed to be overfitting on the training data. So, we add some Dropout layers and employ L1 and L2 regularization to prevent this.

Since the above model overfits the training data, we use a simpler model, with a single convolution layer with ReLU activation having 128 channels as well as a MaxPooling layer. We increase the dense layers to 3 and set their sizes as (256, 128, 20) in order of increasing depth. Adding regularization and dropout layers after MaxPooling and before the last dense layer, we achieve optimum accuracy. The substantial increase in accuracy (and decrease in overfit) is demonstrated in Fig. 2.

Fig. 3 presents a schematic diagram of the optimized CNN described above.

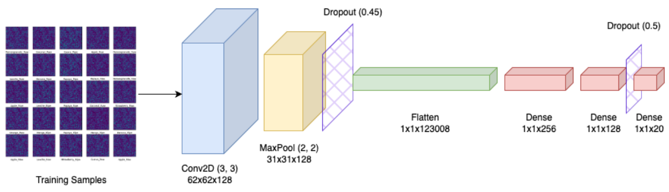


Fig. 3: Optimized CNN

VI. Results and Final Workflow

A. Voting Classifier Workflow

We use the Voting Classifier ensemble method to create our final prediction set. The workflow given in Figure 4 shows the methodology adopted in training the Voting Classifier for making the predictions:

This adopted methodology gives a **84.541%** accuracy on the public leaderboard and a **82.5%** accuracy on the private leaderboard.

B. CNN Workflow

The workflow given in Figure 5 shows the methodology adopted to train a Convolutional Neural Network for predicting the classes in the test data-set:

This adopted methodology gives a **79.8%** accuracy on the public leaderboard and a **84.5%** accuracy on the private leaderboard in the Kaggle Contest for this project.

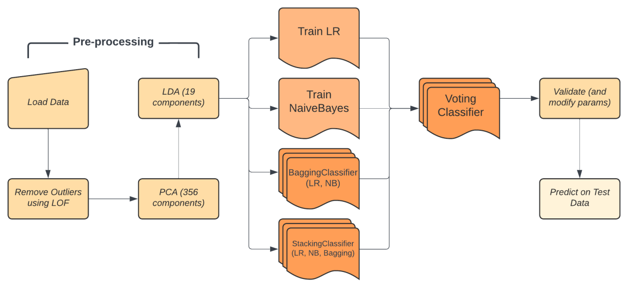


Fig. 4: Voting Classifier Workflow

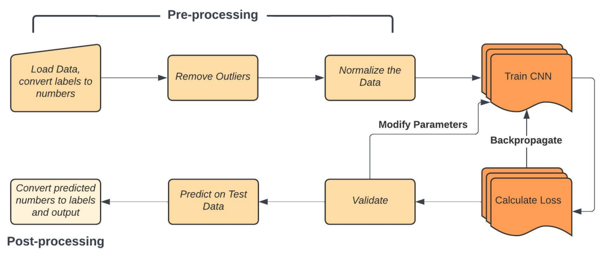


Fig. 5: CNN Workflow

VII. Literature Review

The Literature Review as asked contains a brief description of all the methods which we have used in this project along with their applications. They are listed as follows:

1) Logistic Regression: Logistic regression is a statistical method used to model and analyze the relationship between a binary dependent variable and one or more independent variables. Applications of this technique include: Predicting the likelihood of a customer making a purchase based on their demographic information and Identifying the factors that contribute to the success or failure of a marketing campaign.

2) k-Nearest Neighbors: K nearest neighbors is a machine learning algorithm used for classification and regression analysis that identifies the k closest data points in a dataset to a new input, and then classifies or predicts the output based on the majority of those k neighbors. Three applications of k nearest neighbors include: Identifying fraudulent credit card transactions based on the similarity of the transaction to past fraudulent transactions.

3) Naives Bayes Classifier: Naive Bayes classifier is a probabilistic machine learning algorithm that uses Bayes' theorem to predict the likelihood of a certain class based on the presence of certain features in the input data. Applications of naive Bayes classifier include: Spam filtering in emails by predicting the likelihood of an email being spam based on certain keywords and Document classification by predicting the topic of a document based on the presence of certain words.

4) Decision Tree Classifier: Decision tree classifier is a machine learning algorithm that builds a tree-like model to make decisions based on the features of the input data. Applications of decision tree classifier include Credit risk assessment by predicting the likelihood of a loan being repaid based on certain borrower characteristics and medical diagnosis by predicting the likelihood of a certain disease based on symptoms and patient characteristics

5) Random Forest Classifier: Random forest is a powerful machine learning algorithm that creates an ensemble of decision trees, where each tree is trained on a random subset of the data and features. It works by aggregating the predictions of individual trees to make a final prediction. An application of random forest is in medical diagnosis, where it is used to classify patients as either having a particular disease or not.

6) Gradient Boosting Classifier: Gradient Boosting is a machine learning algorithm that creates an ensemble of decision trees by iteratively fitting new trees to the residual errors of the previous trees. It works by minimizing a loss function, which measures the difference between the predicted and actual values. An application of Gradient Boosting is in predicting customer churn in telecom companies, where it is used to identify customers who are likely to switch to a different provider.

7) ADABoost Classifier: Adaboost is a popular machine learning algorithm that combines multiple weak classifiers to form a strong classifier. It works by iteratively adjusting the weights of misclassified samples to improve the overall performance of the classifier. An application of Adaboost is in face detection, where it is used to classify images as either containing a face or not. Adaboost can learn to detect various features of a face, such as the presence of eyes, nose, and mouth, and combine them to accurately classify images.

8) Bagging Classifier: Bagging (Bootstrap Aggregation) is a machine learning technique that involves training multiple instances of a single model on different subsets of the training data and aggregating their predictions to reduce overfitting. An application of Bagging is in credit risk assessment, where it is used to classify loan applicants as either high or low risk.

9) Stacking and Voting Classifier: Stacking is a machine learning technique that involves training multiple models and using their predictions as inputs to a meta-model, while Voting classifier is a technique that combines the predictions of multiple models to make a final prediction based on majority voting. They are used in image classification and predicting customer satisfaction.

10) Clustering: KMeans is a clustering technique that groups data points into k clusters based on their similarity in Euclidean space, Fuzzy c-means is a soft clustering technique that assigns each data point a degree of membership to each cluster based on its similarity, Mean Shift is a density-based clustering technique that identifies clusters as high-density regions in the data, and DBSCAN is a density-based clustering technique that groups data points into clusters based on their density and proximity to each other. An application of clustering is in customer segmentation, where clustering techniques can be used to group customers based on their behavior, preferences, or demographics. This can help businesses to tailor their marketing strategies, improve customer satisfaction, and increase revenue.

11) Outlier Detection: Local Outlier Factor (LoF) is an outlier detection technique that identifies data points with low local densities compared to their neighbors, while Isolation Forests is a tree-based technique that isolates outliers by repeatedly partitioning the data and identifying the most ”isolated” data points. An application of outlier detection is in fraud detection, where outlier detection techniques can be used to identify suspicious transactions or activities that deviate from normal behavior.

12) Dimensionality Reduction: Principal Component Analysis (PCA) is a dimensionality reduction technique that transforms high-dimensional data into a lower-dimensional space while preserving most of the original variance, while Linear Discriminant Analysis (LDA) is a supervised dimensionality reduction technique that maximizes the separation between different classes in the data. Techniques like these are used for face recognition and in image compression.

VIII. References

1) Article on Voting Classifiers
2) Scikit-Learn Module on Logistic Regression
3) Scikit-Learn Module on Random Forest Classifiers
4) Article on Random Forest Classifiers and its Hyperparameters
5) Article on Gradient Boosting Classifier
6) Article on Local Outlier Factor (LoF)
7) Article on Fuzzy C-Means Clustering using Python
8) Article on Transformers for Image Classification
9) Article on Random State in Sklearn
10) TensorFlow's CNN demo
11) Article on strategies to reduce overfitting in CNNs