# **Statistical/Machine Learning and Data Mining Project Report**

**Student Name: Deepesh Ramegowda**

**Student ID: 201905078**

**Project Name: Classification of Fruits and Vegetables**

Original Dataset Repository: <https://www.kaggle.com/moltean/fruits>

The actual Dataset Consists of 82213 images of 120 Categories with Image Size 100\* 100

In my project I have chosen 22846 images of 46 categories which is resized for 30\*30 using the resize script that would generate my test and train samples in \*\*\*\*\*\*.npy format

The dataset in my experiment is reduced for 46 categories due to computational issues.

**Solutions on the Dataset set Provided:**

1. The solutions that I have worked for my Dataset is as Below:
2. CNN with K fold Cross Validation with 10 Splits
3. Accuracy Rate of LDA, QDA, Gaussian, KNN-1, KNN-5, KNN-10 WITH K Fold cross validation
4. Accuracy Rate of LDA, QDA, Gaussian, KNN-1, KNN-5, KNN-10 WITH K Fold cross validation when applied with PCA
5. K Means Clustering with Elbow and Cluster based on output from Elbow method.
6. Random Forest
7. Support Vector Machine
8. Hierarchical Clustering
9. XGBOOST Algorithm
10. Tried predicting on Celebrities Faces over 100 categories – CNN
11. Running a simple Logistic regression on Two labels (Binary Classification)

**Why the image was Resized only for 30\* 30?**

The Script for resize was done generally for reducing the image size and also to maintain uniform image size Hence Due to computational consideration which was experimented which lead to conclusion of 30\* 30 would be the better approach for working on the Data Set that will be : 30\*30\*3 = 2700 Pixels

Ideally the script would convert the images in to \*\*\*\*.npy files which would be better for loading the images and processing rather than reading the images all the time which would reduce the time on Data Pre-processing stage. The output of the script will help create labels.npy as well as data.npy in the respective folder as named in the script.

We also Tried with 60\* 60 resizing since the computation will be 60\*60\*3 = 10800 px was more. So ideally decided that it would be a better approach for 30\*30

**What are the steps involved in Data pre-processing ?**

The actual Train and Test folder had an image of 120 categories like [Apple1, AppleType2, AppleType3, …. Banana , Lemon ,Lemon Meyer, …....]

The actual dataset that I have used before resizing is in the Drive Link:

https://drive.google.com/file/d/1Z-T5MxMD5wG5i3n7MVPNLGhC-dk9atKB/view?usp=sharing

|  |
| --- |
| [ A high-quality, dataset of images containing fruits and vegetables. The following fruits and vegetables are included: Apples (different varieties: Crimson Snow, Golden, Golden-Red, Granny Smith, Pink Lady, Red, Red Delicious), Apricot, Avocado, Avocado ripe, Banana (Yellow, Red, Lady Finger), Beetroot Red, Blueberry, Cactus fruit, Cantaloupe (2 varieties), Carambula, Cauliflower, Cherry (different varieties, Rainier), Cherry Wax (Yellow, Red, Black), Chestnut, Clementine, Cocos, Dates, Eggplant, Ginger Root, Granadilla, Grape (Blue, Pink, White (different varieties)), Grapefruit (Pink, White), Guava, Hazelnut, Huckleberry, Kiwi, Kaki, Kohlrabi, Kumsquats, Lemon (normal, Meyer), Lime, Lychee, Mandarine, Mango (Green, Red), Mangostan, Maracuja, Melon Piel de Sapo, Mulberry, Nectarine (Regular, Flat), Nut (Forest, Pecan), Onion (Red, White), Orange, Papaya, Passion fruit, Peach (different varieties), Pepino, Pear (different varieties, Abate, Forelle, Kaiser, Monster, Red, Williams), Pepper (Red, Green, Yellow), Physalis (normal, with Husk), Pineapple (normal, Mini), Pitahaya Red, Plum (different varieties), Pomegranate, Pomelo Sweetie, Potato (Red, Sweet, White), Quince, Rambutan, Raspberry, Redcurrant, Salak, Strawberry (normal, Wedge), Tamarillo, Tangelo, Tomato (different varieties, Maroon, Cherry Red, Yellow), Walnut.] |

The Down Sampling was done considering [ Apple, Banana, Lemon] in both Testing and Training as show in image

|  |
| --- |
| Apples , Apricot, Avocado, Avocado ripe, Banana , Beetroot Red, Blueberry, Cactus fruit, Cantaloupe , Carambula, Cauliflower, Cherry, Clementine, Cocos, Dates, Eggplant, Ginger , Granadilla, Grape , Guava,Kiwi, Kaki, Kohlrabi, Kumsquats, Lemon (normal), Lime, Lychee, Mandarine, Mango , Mangostan, Maracuja, Melon Piel de Sapo, Mulberry, Nectarine (Regular), Nut Onion , Orange, Papaya, Passion fruit, Peach, Pepino, Pear , Pepper , Physalis, Pineapple (normal, Pitahaya Red, Plum , Pomegranate, Pomelo Sweetie, Potato (Red, Sweet, White), Quince, Rambutan, Raspberry, Redcurrant, Salak, Strawberry , Tamarillo, Tangelo, Tomato), Walnut.] |

The Resize.py Script that is uploaded will reshape the images to 30\*30 and save the Image data and the labels in the folder with \*\*\*\*.npy Extension which will be a 4D matrix of 22846\*30\*30\*3 and Its then applied to CNN. The Resize is done for the Testing samples which also included 46 Test Data Categories.

The Figure below shows the Information of Test and Training Images

Figure: Training Dataset with 46 categories:

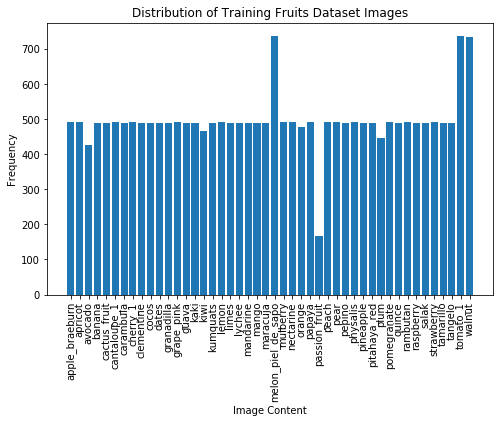
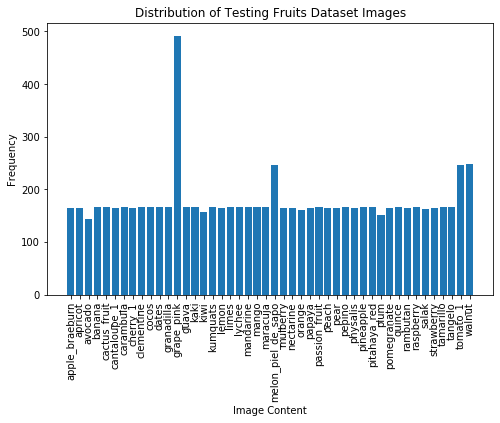


Figure: Testing Dataset with 46 Testing Category



**Experiment 1**:  **CNN with K fold Cross Validation with 10 Splits**

|  |
| --- |
| The Total Training data that was:  Train Data (22846, 2700)  Training Labels (22846,)  Testing Data (8119, 2700)  Testing Labels (8119,) |

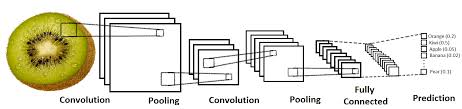


Image Source: <https://ryan-kttam.github.io/fruit_image_recognition/>

**What was done in CNN?**

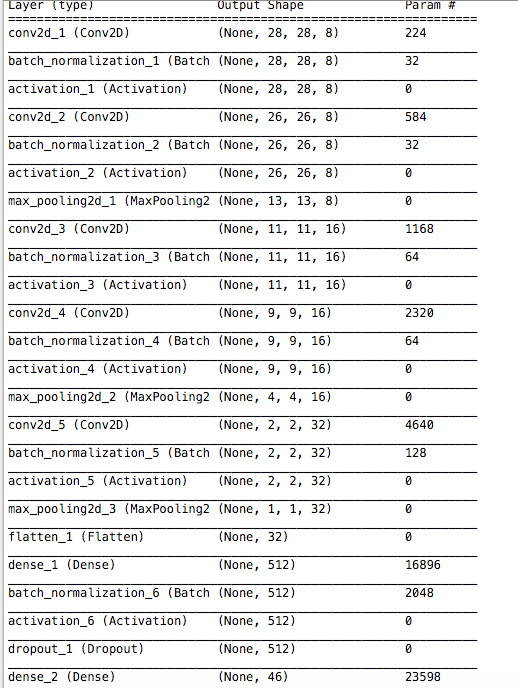
As the CNN is best known for Image classification. I choose to work on CNN to know what model will be best suited for my Experiment to predict the Categories.

The KFold validation is applied with KFold = 10 helps and created a 5 Convolution layer with activation function like Relu and Optimizers = Adam() , MaxPooling , batch size and Epochs. There was lot of Experiments that was done for choosing the right model by adding layers and trying it tuning with the Hyperparameters

The CNN design with proper tuning hyper parameters which includes many layers and activation functions that are not accurate if we implement the architecture with many layers and changing the activation will create more generalised decision boundaries .

The activation function that defines the output of the neuron based of the given set of Inputs here Relu works for this Model .

**The Model was Created with below summary :**



**What was the Experiment Result?**

The K Fold Cross Validation was done with 10 folds which helped for getting the train and the test split and experimented with the actual testing data starting to increase the accuracy of the evaluation on the Model.

The Results are as show below for 10 Splits:

Split One:

|  |  |  |  |
| --- | --- | --- | --- |
| **Split One:**  Epoch 1/4  20547/20547 [==============================] - 27s 1ms/step - loss: 1.0521 - accuracy: 0.7020 - val\_loss: 0.2560 - val\_accuracy: 0.9282  Epoch 2/4  20547/20547 [==============================] - 25s 1ms/step - loss: 0.2271 - accuracy: 0.9314 - val\_loss: 0.1388 - val\_accuracy: 0.9478  Epoch 3/4  20547/20547 [==============================] - 28s 1ms/step - loss: 0.1419 - accuracy: 0.9556 - val\_loss: 0.4501 - val\_accuracy: 0.8225  Epoch 4/4  20547/20547 [==============================] - 29s 1ms/step - loss: 0.1150 - accuracy: 0.9654 - val\_loss: 0.3316 - val\_accuracy: 0.9365  8119/8119 [==============================] - 2s 272us/step    Test loss: 0.12322329680887756  Test Accuracy 0.9579997658729553 | |  | | --- | |  | |  | |

|  |  |  |
| --- | --- | --- |
| **Split Two:**    Epoch 1/4  20547/20547 [==============================] - 27s 1ms/step - loss: 1.0621 - accuracy: 0.6984 - val\_loss: 0.2220 - val\_accuracy: 0.9291  Epoch 2/4  20547/20547 [==============================] - 25s 1ms/step - loss: 0.2165 - accuracy: 0.9351 - val\_loss: 0.1576 - val\_accuracy: 0.9304  Epoch 3/4  20547/20547 [==============================] - 25s 1ms/step - loss: 0.1445 - accuracy: 0.9555 - val\_loss: 0.1047 - val\_accuracy: 0.9661  Epoch 4/4  20547/20547 [==============================] - 26s 1ms/step - loss: 0.0810 - accuracy: 0.9759 - val\_loss: 0.2302 - val\_accuracy: 0.9200  8119/8119 [==============================] - 2s 267us/step    Test loss: 0.3751034105607142  Test Accuracy 0.8770784735679626 | |  | | --- | |  | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Split Three:**  Epoch 1/4  20562/20562 [==============================] - 29s 1ms/step - loss: 1.0958 - accuracy: 0.6922 - val\_loss: 0.2358 - val\_accuracy: 0.9343  Epoch 2/4  20562/20562 [==============================] - 30s 1ms/step - loss: 0.2345 - accuracy: 0.9273 - val\_loss: 0.1088 - val\_accuracy: 0.9637  Epoch 3/4  20562/20562 [==============================] - 26s 1ms/step - loss: 0.1211 - accuracy: 0.9635 - val\_loss: 0.0641 - val\_accuracy: 0.9807  Epoch 4/4  20562/20562 [==============================] - 29s 1ms/step - loss: 0.0959 - accuracy: 0.9717 - val\_loss: 0.0684 - val\_accuracy: 0.9781  8119/8119 [==============================] - 2s 287us/step    Test loss: 0.1348653180603899  Test Accuracy 0.96292644739151 | |  | | --- | |  | |  | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Split Four:**  Epoch 1/4  20562/20562 [==============================] - 30s 1ms/step - loss: 1.0540 - accuracy: 0.6987 - val\_loss: 0.1637 - val\_accuracy: 0.9426  Epoch 2/4  20562/20562 [==============================] - 30s 1ms/step - loss: 0.2180 - accuracy: 0.9335 - val\_loss: 0.1871 - val\_accuracy: 0.9291  Epoch 3/4  20562/20562 [==============================] - 29s 1ms/step - loss: 0.1262 - accuracy: 0.9612 - val\_loss: 0.1188 - val\_accuracy: 0.9593  Epoch 4/4  20562/20562 [==============================] - 26s 1ms/step - loss: 0.0904 - accuracy: 0.9718 - val\_loss: 0.0547 - val\_accuracy: 0.9803  8119/8119 [==============================] - 2s 228us/step    Test loss: 0.14359946221406572  Test Accuracy 0.958246111869812 | |  | | --- | |  | |  | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Split Five:**  Epoch 1/4  20562/20562 [==============================] - 30s 1ms/step - loss: 1.0469 - accuracy: 0.6964 - val\_loss: 0.2716 - val\_accuracy: 0.9098  Epoch 2/4  20562/20562 [==============================] - 43s 2ms/step - loss: 0.2303 - accuracy: 0.9282 - val\_loss: 0.1061 - val\_accuracy: 0.9575  Epoch 3/4  20562/20562 [==============================] - 35s 2ms/step - loss: 0.1276 - accuracy: 0.9606 - val\_loss: 0.1004 - val\_accuracy: 0.9641  Epoch 4/4  20562/20562 [==============================] - 24s 1ms/step - loss: 0.0905 - accuracy: 0.9721 - val\_loss: 0.0808 - val\_accuracy: 0.9707  8119/8119 [==============================] - 2s 261us/step    Test loss: 0.12230247112139704  Test Accuracy 0.9604631066322327 | |  | | --- | |  | |  | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Split Six:**  Epoch 1/4  20563/20563 [==============================] - 31s 2ms/step - loss: 1.0228 - accuracy: 0.7105 - val\_loss: 0.5213 - val\_accuracy: 0.8257  Epoch 2/4  20563/20563 [==============================] - 28s 1ms/step - loss: 0.1877 - accuracy: 0.9445 - val\_loss: 0.5796 - val\_accuracy: 0.8217  Epoch 3/4  20563/20563 [==============================] - 28s 1ms/step - loss: 0.1027 - accuracy: 0.9687 - val\_loss: 0.5952 - val\_accuracy: 0.8366  Epoch 4/4  20563/20563 [==============================] - 24s 1ms/step - loss: 0.0746 - accuracy: 0.9767 - val\_loss: 0.3826 - val\_accuracy: 0.8922  8119/8119 [==============================] - 2s 240us/step    Test loss: 0.14565199053757338  Test Accuracy 0.9528266787528992  dict\_keys(['val\_loss', 'val\_accuracy', 'loss', 'accuracy']) | |  | | --- | |  | |  | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Split Seven:**  20565/20565 [==============================] - 30s 1ms/step - loss: 1.0224 - accuracy: 0.7170 - val\_loss: 0.4646 - val\_accuracy: 0.8400  Epoch 2/4  20565/20565 [==============================] - 28s 1ms/step - loss: 0.2027 - accuracy: 0.9387 - val\_loss: 0.2477 - val\_accuracy: 0.9110  Epoch 3/4  20565/20565 [==============================] - 26s 1ms/step - loss: 0.1197 - accuracy: 0.9656 - val\_loss: 0.2348 - val\_accuracy: 0.9163  Epoch 4/4  20565/20565 [==============================] - 25s 1ms/step - loss: 0.0834 - accuracy: 0.9746 - val\_loss: 0.2657 - val\_accuracy: 0.9062  8119/8119 [==============================] - 2s 258us/step    Test loss: 0.16885439210952663  Test Accuracy 0.9524571895599365 | |  | | --- | |  | |  | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Split Eight:**  Epoch 1/4  20567/20567 [==============================] - 31s 1ms/step - loss: 1.0838 - accuracy: 0.6879 - val\_loss: 0.4786 - val\_accuracy: 0.8653  Epoch 2/4  20567/20567 [==============================] - 27s 1ms/step - loss: 0.2330 - accuracy: 0.9278 - val\_loss: 0.3395 - val\_accuracy: 0.8780  Epoch 3/4  20567/20567 [==============================] - 24s 1ms/step - loss: 0.1345 - accuracy: 0.9595 - val\_loss: 0.2078 - val\_accuracy: 0.9289  Epoch 4/4  20567/20567 [==============================] - 24s 1ms/step - loss: 0.1036 - accuracy: 0.9671 - val\_loss: 0.2921 - val\_accuracy: 0.9065  8119/8119 [==============================] - 2s 236us/step    Test loss: 0.14473194961668895  Test Accuracy 0.9497475028038025 | |  | | --- | |  | |  | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Split Nine:**  Epoch 1/4  20569/20569 [==============================] - 30s 1ms/step - loss: 1.0107 - accuracy: 0.7092 - val\_loss: 0.2426 - val\_accuracy: 0.9205  Epoch 2/4  20569/20569 [==============================] - 30s 1ms/step - loss: 0.2179 - accuracy: 0.9337 - val\_loss: 0.2198 - val\_accuracy: 0.9310  Epoch 3/4  20569/20569 [==============================] - 25s 1ms/step - loss: 0.1238 - accuracy: 0.9625 - val\_loss: 0.0648 - val\_accuracy: 0.9767  Epoch 4/4  20569/20569 [==============================] - 26s 1ms/step - loss: 0.0958 - accuracy: 0.9703 - val\_loss: 0.0935 - val\_accuracy: 0.9679  8119/8119 [==============================] - 2s 286us/step    Test loss: 0.11512082265318575  Test Accuracy 0.9653898477554321 | |  | | --- | |  | |  | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Split Ten :**  Epoch 1/4  20570/20570 [==============================] - 32s 2ms/step - loss: 1.0712 - accuracy: 0.6961 - val\_loss: 0.3396 - val\_accuracy: 0.8985  Epoch 2/4  20570/20570 [==============================] - 29s 1ms/step - loss: 0.2365 - accuracy: 0.9290 - val\_loss: 0.3749 - val\_accuracy: 0.8937  Epoch 3/4  20570/20570 [==============================] - 27s 1ms/step - loss: 0.1313 - accuracy: 0.9601 - val\_loss: 0.2440 - val\_accuracy: 0.9389  Epoch 4/4  20570/20570 [==============================] - 25s 1ms/step - loss: 0.1022 - accuracy: 0.9683 - val\_loss: 0.3453 - val\_accuracy: 0.9205  8119/8119 [==============================] - 2s 239us/step    Test loss: 0.099896637479785  Test Accuracy 0.9616947770118713 | |  | | --- | |  | |  | |

**Conclusion:**

The CNN could predict a very good accuracy of 96% for the Training set as well as Testing set.

**Experiment 2:**

The experiment 2 was done with K Fold Validation K=10 over LDA, QDA, GaussianNB , KNN =1,5,10

These Experiments are done on the Actual Dataset **without PCA.**

LDA GAUSSIAN KNN =1,5,10 gives the best result for the dataset. But QDA is not suitable for the experiment as it has an accuracy of 24%

|  |
| --- |
| The LDA MEAN ERROR RATE 0.0015754923413566738 GAUSSIAN MEAN ERROR RATE 0.05724288840262581 QUAD mean ERROR RATE 0.753085339168490  KNN ONE mean ERROR RATE 8.7527352297593e-05 KNN TWO mean ERROR RATE 0.0007439824945295405 KNN TEN mean ERROR RATE 0.002013129102844639 |

|  |
| --- |
| LDA MEAN ACCURACY RATE 0.9984245076586434 GAUSSIAN MEAN ACCURACY RATE 0.9427571115973743 QUAD mean ACCURACY RATE 0.24691466083150987  KNN ONE mean ACCURACY RATE 0.9999124726477024 KNN TWO mean ACCURACY RATE 0.9992560175054706 KNN TEN mean ACCURACY RATE 0.9979868708971553 |

The Graphs for Error Rate and Accuracy Rate is Follows:

|  |
| --- |
|  |

|  |
| --- |
|  |

The KNN Consistently performs best for the Experiment, This would have been further tested for KNN value = 30 neighbours. Here I was not able to do since the Time KNN took for the computation was very high and the IDE was crashed and the complete code had to be redone

Note: The validations are not done here on the actual testing data rather the Testing is done on the Test set of Stratified Shuffle Split.

As per the LDA also performs the best for the Dataset LDA maximize the Class separability. Its basically works for classification and Dimensionality reduction techniques. That outperforms in my Dataset since LDA takes Labels into Account and objective of LDA would be to keep the datapoints of the same class more compact and have a clean class separability. LDA is ideal for Classification Problems

Gaussian NB would also be in the competition for choosing the Right predictions, according to Gaussian uses Bayes Learning techniques and observe over the training Data It’s a Standard Techniques used for Classification Problems. The Gaussian works well for multiple Class prediction problems.

QDA will have the least Accuracy when compared with LDA KNN and Gaussian Techniques. The following might be the reason for QDA to perform bad on my Dataset

**Experiment 3:**

The experiment 2 was done with K Fold Validation K=10 over LDA, QDA, GaussianNB , KNN =1,5,10

By applying PCA Techniques with 2 Components:

These Experiments are done on the Actual Dataset **with PCA = 2 Components**

|  |
| --- |
| LDA MEAN ERROR RATE 0.6931728665207877  GAUSSIAN MEAN ERROR RATE: 0.6069584245076587  QUAD mean ERROR RATE 0.5558424507658642  KNN ONE mean ERROR RATE 0.31737417943107216  KNN TWO mean ERROR RATE 0.2985557986870897  KNN TEN mean ERROR RATE: 0.3066520787746171 |

|  |
| --- |
| LDA MEAN ACCURACY RATE 0.3068271334792122  GAUSSIAN MEAN ACCURACY RATE : 0.39304157549234137  QUAD mean ACCURACY RATE 0.44415754923413575  KNN ONE mean ACCURACY RATE 0.6826258205689278  KNN TWO mean ACCURACY RATE 0.7014442013129103  KNN TEN mean ACCURACY RATE : 0.693347921225383 |

|  |
| --- |
|  |

|  |
| --- |
|  |

After applying PCA The accuracy of LDA Gaussian KNN =1,5,10 is l**ow when compared with actually data without PCA**

Since it’s a PCA is Known for Dimensionality reduction Techniques, Feature extraction technique that convert possible set of corelated variables using orthogonal transformation. The Experiment of PCA on My Dataset has be worse. This we generally call it as **Curse of Dimensionality reduction** Techniques.

The data in high dimensionality space are very hard to occur in Lower dimension space, Due to Higher dimension the model gets sparse, Higher dimensionality space causes problem in clustering and increase model complexity.

PCA on Dimensionality reduction we might be facing Data loss for in our techniques.

**Note:** We are observing the increase in accuracy for QDA after Applying PCA with 2 Components

**The feature Extraction Techniques that converts corelates possible datapoints to uncorrelated variables .**

**Experiment has been done with PCA N\_COMPONENTS = 30 and check the accuracy Rate and have we see Exceptional Results:**

|  |
| --- |
| LDA MEAN ERROR RATE 0.08205689277899345  GAUSSIAN MEAN ERROR RATE: 0.11067833698030635  QUAD mean ERROR RATE 0.0003063457330415755  KNN ONE mean ERROR RATE 0.0001312910284463895  KNN TWO mean ERROR RATE 0.0006564551422319475  KNN TEN mean ERROR RATE : 0.002013129102844639 |

|  |
| --- |
| LDA MEAN ACCURACY RATE 0.9179431072210067  GAUSSIAN MEAN ACCURACY RATE: 0.8893216630196937  QUAD mean ACCURACY RATE 0.9996936542669586  KNN ONE mean ACCURACY RATE 0.9998687089715537  KNN TWO mean ACCURACY RATE 0.9993435448577681  KNN TEN mean ACCURACY RATE 0.9979868708971553 |

|  |
| --- |
| **PCA** n\_components = 30  [0.29770357 0.13717681 0.11469967 0.04346346 0.03541376 0.0316553  0.02167677 0.01993218 0.01693086 0.016269 0.01427941 0.01097891  0.01000449 0.00917465 0.0077526 0.0068152 0.00629201 0.00603767  0.0055502 0.00527222 0.00489224 0.00477775 0.00426018 0.00410863  0.00376846 0.00351517 0.0032533 0.00309742 0.00297786 0.0027146 ] |

|  |
| --- |
|  |

|  |
| --- |
|  |

**Conclusion :**

**Now with PCA n\_components = 30. We get exceptional output when compared with n\_components = 2 Now we appreciate PCA as Dimensionality reduction techniques.**

**Experiment 4:** I tried to work on KMeans Clustering for classification problem

We applied KMeans Clustering for the Dataset, I have chosen **PCA n\_components = 2** and try to observe the Elbow Method knowing that in background I have 46 labels to be clustered. but the results were not so appreciable. I tried with various n\_components for PCA value but the results are not so approachable to further continue to choose this classifier.

The clusters was purposefully put for 46 to clusters to check the cluster which is not correct.

|  |
| --- |
| Elbow method for 130 |

|  |
| --- |
|  |

**Why K Means Clustering was not the suitable solution for this dataset?**

Clustering is and unsupervised machine learning topics the clustering process organises objects into groups which are similar in some way and they try to form cluster of same kind. The task here would cluster similar objects in general it clusters all the fruits as Fruits rather than clustering apple and banana as two clusters.

**Conclusion**: The KMeans clustering is a unsupervised Machine Learning Algorithm applying classification task would not be appropriate to classify the object

**Experiment 5: Random Forest Classifier**

The Experiment was done using Random Forest Classifier and the prediction are as follows:

The parameter that we have used is as follows:

|  |
| --- |
| classifier = RandomForestClassifier(n\_estimators = 70, criterion = 'entropy', random\_state = 0)  classifier.fit(Xdata, ylables)  y\_pred = classifier.predict(Xdata\_Testing)  Accuracy for Random Forest is 0.9746274171696022 |

|  |
| --- |
| classifier = RandomForestClassifier(n\_estimators = 200, criterion = 'entropy', random\_state = 0)  classifier.fit(Xdata, ylables)  y\_pred = classifier.predict(Xdata\_Testing)  Accuracy for Random Forest is 0.9762285995812292 |

|  |
| --- |
| pca = PCA(n\_components = 30)  Xdata = pca.fit\_transform(Xdata)  Xdata\_Testing = pca.transform(Xdata\_Testing)    classifier = RandomForestClassifier(n\_estimators = 200, criterion = 'entropy', random\_state = 0)  classifier.fit(Xdata, ylables)    # Predicting the Test set results  y\_pred = classifier.predict(Xdata\_Testing)  Accuracy for Random Forest is 0.9477768198053947 |

Random forest works on the principal of Ensemble Learning:

The Ensemble Learning will work on the basic of Bagging and Boosting.: The decision of the tree is not based on one dependent model . Rather it’s a collective voting of many classifiers as shown in the image below This would be the best techniques that can be used for based on hard voting and soft voting concepts

|  |
| --- |
|  |

Image Source : <https://towardsdatascience.com/random-forest-and-its-implementation-71824ced454f>

**Experiment 6**: Hierarchical Clustering

The dataset was tested on Hierarchical Clustering Technique and tried to plot the Dendogram first

The output of the Dendogram for all the 46 labels is as shown in the Image The dendogram is generated with size 300,50 on Alienware Lab system and the same code . This could take lot of time to generate the output.

I was not able to achieve the colour for all the clusters:

|  |
| --- |
|  |

The cluster is created with cluster=46 and the prediction are printed without applying PCA.

The changes are made with PCA components= 30 and with components =2 and the prediction are printed in the code this would be enabled in the code.

The experiment was done with PCA =2

|  |
| --- |
| hc = AgglomerativeClustering(n\_clusters = 46, affinity = 'euclidean', linkage = 'ward')  y\_hc = hc.fit\_predict(Xdata)    plt.scatter(Xdata[:,0], Xdata[:,1],c=hc.labels\_.astype(float)) |

**Experiment 7: Support Vector Machine:**

The Experiment is done using Support Vector Machine

The Basic implementation of SVM is done in the Project to see what would be prediction rate of SVM The result are as follows **with PCA components 2 :**

|  |
| --- |
| The accuracy for SVM with PCA = 2 will be: 0.11848749846040153  The accuracy for SVM with PCA = 20 will be: 0.11134376154698855 |

The support vector machine is suited for both classification and regression problems. which creates linear boundary by maximizing the distance between the class however due to PCA the SVM accuracy is not appreciated due to dimensionality reduction technique. (Curse of Dimensionality reduction)

**Experiment 8:**  XGBOOST

The recent powerful Machine Learning Algorithm, Specially works with Large Dataset and the higher performance execution speed I have just tried to implement the basic of this XGBoost to understand the speed and the performance of this algorithm. Since the major challenge that I faced in the overall project was the speed and time computation taken by each algorithm

It's just a try to understand what is making this XGBOOST so famous in terms of speed and performance

Citation:

<https://xgboost.readthedocs.io/en/latest/>

As per the sources the XGBOOST is a optimized distributed gradient boosting library designed to be highly efficient, flexible and portable. It implements machine learning algorithms under the [Gradient Boosting](https://en.wikipedia.org/wiki/Gradient_boosting) framework. XGBoost provides a parallel tree boosting.

Installation procedure: These steps were part of the of the installation procedure .

<https://xgboost.readthedocs.io/en/latest/build.html>

Install Homebrew:

|  |
| --- |
|  |

Cloning of XGBOOST Setup:

|  |
| --- |
| git clone --recursive <https://github.com/dmlc/xgboost> |

I tried working with XGboost to know what will be the performance and the prediction that will be improved on my Dataset. Unfortunately this didn't work out and it took lot of time and was with some error. I tried fixing this issue but hard luck could not find any this was basically installation setup issue in my Mac machine.

**Experiment 9:** The initial idea for Graduate project was to work on faces Dataset specially with twin images to know what the performance of CNN .

Later due to lack of dataset on twins I shifted the Idea to Celebrities Faces from Pin interest:

The Original Dataset link:

https://drive.google.com/open?id=1voGvzJSXGTK6mRQ22qLyLz\_-v0nH7J7H

The Description of the Dataset and the Data is as below:

|  |
| --- |
|  |

The total sample was 10770 which is of random sizes.

The idea was to pick the 100 celebrities faces and predict the name of the Celebrities:

The idea of resizing the Images started here since I could see lot of image with different dimension I ideally synced with Derek to work on the Faces we were successfully able to reshape the image for 100\*100 \*3 and try CNN on the same.

The data is augmented with varying rotation angles this was challenging to predict the class lables accurately

**Why the image was Resized only for 100\* 100?**

The Script for resize was done generally for reducing the image size and also to maintain uniform image size Hence Due to computational consideration which was experimented which lead to conclusion of 100\* 100 would be the better approach for working on the Data Set that will be : 100\*100\*3 = 30000 Pixels

Ideally the script would convert the images in to \*\*\*\*.npy files which would be better for loading the images and processing rather than reading the images all the time which would reduce the time on Data Pre-processing stage. The output of the script will help create labels.npy as well as data.npy in the respective folder as named in the script.

The initial Dataset with 100 celebrities was chosen and the below is the names of those celebrities (Labels)

|  |
| --- |
|  |

**What happened with the experiment on Faces ?**

The Kfold cross validation was added where K=10 was applied with 1 epoch size since the computation time was very expensive which took almost 5 hours on the experiment and with the accuracy of 10% over the time of 5-6 hours

The actual Test and train results is as below:

|  |
| --- |
| **Split One :**  Epoch 1/1  9649/9649 [==============================] - 250s 26ms/step - loss: 4.3952 - accuracy: 0.1130 - val\_loss: 4.0298 - val\_accuracy: 0.0856  1121/1121 [==============================] - 7s 6ms/step    Test loss: 4.029785704548927  Test Accuracy 0.08563782274723053  **Split Two:**  Epoch 1/1  9662/9662 [==============================] - 255s 26ms/step - loss: 4.3915 - accuracy: 0.1104 - val\_loss: 4.1130 - val\_accuracy: 0.0677  1108/1108 [==============================] - 7s 6ms/step    Test loss: 4.1130400375338665  Test Accuracy 0.06768953055143356  **Split Three:**  Epoch 1/1  9672/9672 [==============================] - 251s 26ms/step - loss: 4.3337 - accuracy: 0.1223 - val\_loss: 4.1019 - val\_accuracy: 0.0783  1098/1098 [==============================] - 7s 6ms/step    Test loss: 4.101912543639025  Test Accuracy 0.07832422852516174  **Split Four:**  Epoch 1/1  9680/9680 [==============================] - 256s 26ms/step - loss: 4.5611 - accuracy: 0.0966 - val\_loss: 3.9529 - val\_accuracy: 0.1110  1090/1090 [==============================] - 6s 6ms/step    Test loss: 3.952909524725118  Test Accuracy 0.11100917309522629  **Split Five:**  Epoch 1/1  9649/9649 [==============================] - 246s 25ms/step - loss: 4.3382 - accuracy: 0.1176 - val\_loss: 3.7002 - val\_accuracy: 0.1204  1121/1121 [==============================] - 7s 6ms/step    Test loss: 3.7002121974696656  Test Accuracy 0.1204281896352768  **Split Six:**  Epoch 1/1  9662/9662 [==============================] - 268s 28ms/step - loss: 4.4746 - accuracy: 0.0987 - val\_loss: 4.5006 - val\_accuracy: 0.0551  1108/1108 [==============================] - 6s 6ms/step    Test loss: 4.500598348004723  Test Accuracy 0.05505415052175522  **Split Seven:**  Epoch 1/1  9662/9662 [==============================] - 268s 28ms/step - loss: 4.4746 - accuracy: 0.0987 - val\_loss: 4.5006 - val\_accuracy: 0.0551  1108/1108 [==============================] - 6s 6ms/step    Test loss: 4.500598348004723  Test Accuracy 0.05505415052175522  **Split Eight:**  Epoch 1/1  9680/9680 [==============================] - 264s 27ms/step - loss: 4.6490 - accuracy: 0.0778 - val\_loss: 4.2262 - val\_accuracy: 0.0872  1090/1090 [==============================] - 6s 6ms/step    Test loss: 4.2262115749744105  Test Accuracy 0.08715596050024033  **Split Nine:**  Epoch 1/1  9686/9686 [==============================] - 254s 26ms/step - loss: 4.5638 - accuracy: 0.0883 - val\_loss: 4.2637 - val\_accuracy: 0.0858  1084/1084 [==============================] - 6s 6ms/step    Test loss: 4.263658967845115  Test Accuracy 0.08579336106777191  **Split Ten:**  Epoch 1/1  9696/9696 [==============================] - 253s 26ms/step - loss: 4.4638 - accuracy: 0.1008 - val\_loss: 3.9234 - val\_accuracy: 0.1015  1074/1074 [==============================] - 6s 6ms/step    Test loss: 3.9234384571373795  Test Accuracy 0.10148975998163223 |

The CNN was trained with 5 Convolution Layers. The model would have been trained better with choosing the best Hyperparameters this accuracy was the best I could achieve since due to computation constraints .the actual facts which I could not make it better is because of the Data Augmentation in the dataset .

**Experiment 11:** This Final Experiment was done to achieve Logistic regression and to implement Logistic regression for two Class labels (Apple ,Apricot) only I could able to do only the actual Logistic regression with two Labels of Data from the actual fruits dataset and the accuracy rate as follows :

|  |
| --- |
| Accuracy of Simple Logistic Regression 0.8871951219512195 |

The Logistic regression is an extension of Linear Regression

**Conclusion**: I was not able to make Simple Logistic Regression.

References:

<https://towardsdatascience.com/>

<https://www.datacamp.com/community/tutorials/convolutional-neural-networks-python#cnn>

<https://medium.com/machine-learning-101/chapter-2-svm-support-vector-machine-theory-f0812effc72>

Thanks Dr Jacob Levman for the course and the topics which gives the overall insights of the Machine Learning concepts.

I would like to thank our TA Derek for supporting all day night to achieve something in the project.