Prediction of Alzheimer Disease from MRI Dataset

Gaurav Khatri (A25318673)
The University of Alabama in Huntsville

CS588 : Intro to Big Data Computing

Date *Nov 28, 2022*

1. Introduction

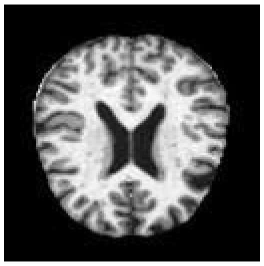
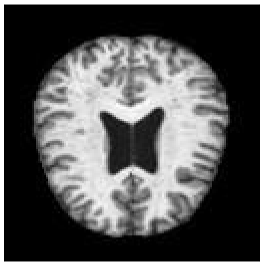
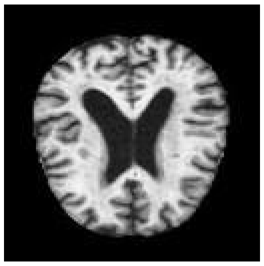
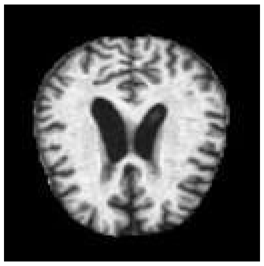
Alzheimer is a progressive neurologic disease that destroys the memory and mental functions of people, especially during old age. Currently no cure exists for this disease, but early detection and management strategies have been found to be effective in slowing down the progression. Hence, data science can be an effective modern tool in helping people identify early onset via careful analysis of medical documents.

In this project, we focus on a Magnetic Resonance Imaging (MRI) dataset of over 6000 patients who have varying severity of the disease. We focus on classifying the data of each patient across 4 categories i.e. No Alzheimer, Very Mild, Mild and Moderate(or High) classes.

2. Methodologies

2.1 Dataset

The dataset has been obtained from Kaggle, which consists of over 6400 samples of MRI images of Alzheimer patients mixed with negative cases. The MRI images thus obtained belong to 4 different classes i.e. No Disease, Very Mild, Mild and Moderate, each with individual resolutions of 128*128 pixels. The data is then downloaded and compiled into a single data source.

 Fig 1 : No Dementia (Class 0)	 Fig 2 : Very Mild Dementia (Class 1)
 Fig 3 : Mild Dementia (Class 2)	 Fig 4 : Moderate Dementia (Class 3)

2.2 Data Processing

In this step we combine the Data Extraction, Cleaning ,Aggregation and Storage steps of the Big Data Analytics Lifecycle. Initially, the data is compiled from the source into a single repository. Then each image is resampled to (60*60*3) image size, in order to maintain uniformity and reduce the data complexity. Then each image is converted to grayscale and flattened to give us a singular representation of 10,800 features per image. Finally all the images are stored in a single Pandas DataFrame in order to make the analysis easier.

2.3 Exploratory Data Analysis

In this step, we aim to understand the data a bit more before moving on towards modeling. Initially we looked at the overall class distribution and found that around 40% of the samples belonged to Class 0. This distribution makes sense, as the percentage of people who have severe cases is much lower than mild cases in real life too.
With over 10000 feature values, it is difficult to understand the overall features for this dataset. Hence we now move towards dimensionality reduction.

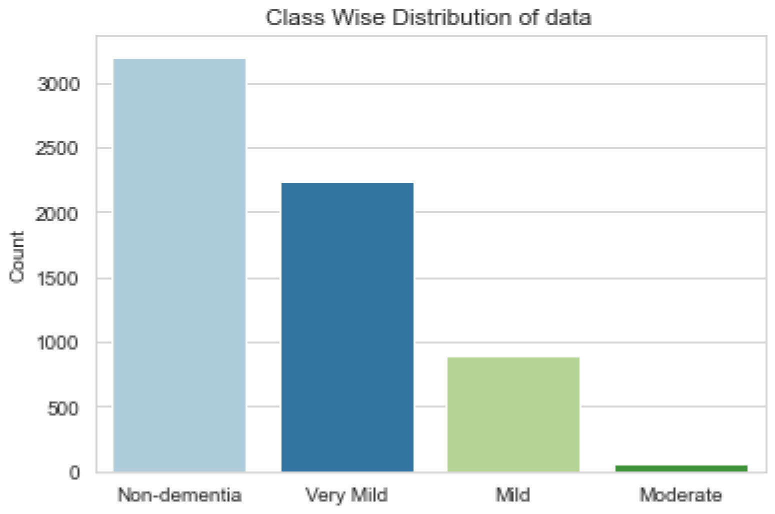


Fig 2.3.1 : Class Wise Distribution of Data Samples

2.4 Dimensionality Reduction
In this step, we construct a sklearn data pipeline in order to Scale the data and then apply dimensionality reduction using PCA. We design the pipeline in such a way that 20% of data samples will be reserved for actual tests at the end and 80% of the data is then used for modeling and cross validation. We tuned our PCA model to attain only 90% of the variance, which resulted in reduced feature size to : 548 dimensions, which is a reduction in dimensions by 20 times.

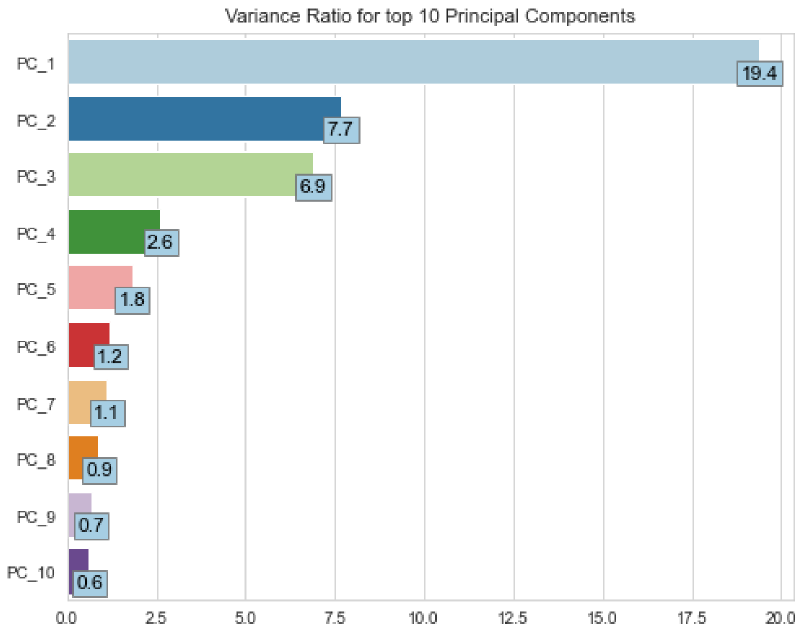


Fig 2.4.1 : Explained Variance Ratio for PCA

2.5 Exploratory Data Analysis after PCA

Since our data was reduced to a much smaller number of dimensions with a fairly better understanding of relative feature importances, different analyses were performed to see the overall data distribution via pairplots.

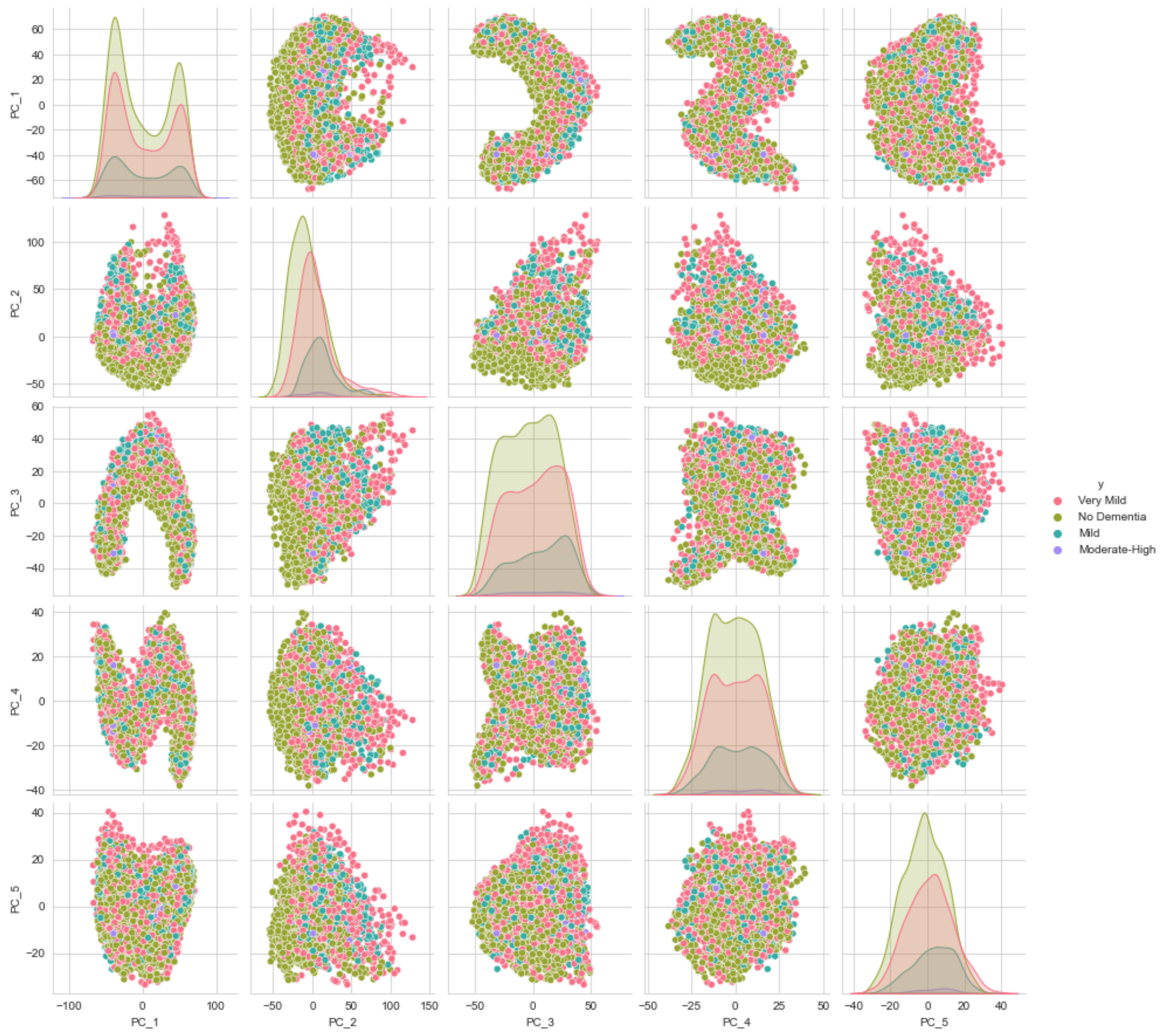


Fig 2.5.1 : Pairplot Distribution of Top 5 PCA components

Upon running the pairplot of top 5 PCA components, we could see that each of the 2d scatter plots were much convoluted. Hence this led to the assumption that the data might not be linearly separable, thus leading to the belief that non-linear classification methods such as SVM (Support Vector Machines) could lead to higher accuracy. Moreover upon seeing the density plots from Fig 2.5.1, it became much clear that each of the features was indeed a good indicator of the class separability.

2.6 Data Modelling

Three different algorithms were applied in this step in order to understand the overall classification problem i.e. Logistic Regression, Random Forest Classifier and Support Vector Machines. Each algorithm was passed through a training pipeline to deduce a robust data model with following steps:

1. Parameter initialization to give a set of parameter space
2. Grid Search Space to search for best solution in the parameter space
3. Cross fold Validation (steps = 10) to find the robust model within the parameter space

The overall search space specification can be tabled as follows:

	Parameters

Logistic Regression	solvers = ['newton-cg','lbfgs', 'liblinear']
	penalty = ['l1','l2','l3']
	c_values = [1.0, 0.1, 0.01,0.001]

Random Forest	n_estimators = [1, 10, 100, 500,1000]
	max_features = ['sqrt', 'log2']

SVM	c_ = [0.01, 0.1, 0.5,1,2,5,10]
	kernel = ['rbf','poly']

3. Results and Analysis

3.1 Logistic Regression

This method resulted in an optimal training accuracy of 78%, subjected to C = 0.01, penalty = L2 and solver = 'newton-cg'. The different cross validation accuracies can be summarized as follows:

Classification Report for Test Data:

	Precision	Recall	F1 Score	Accuracy
Average	0.81	0.77	0.79	0.78

As expected, the overall accuracy for Logistic Regression was below than 80%. This can be attributed to the fact that the underlying data is non-linear and hence less accuracy.

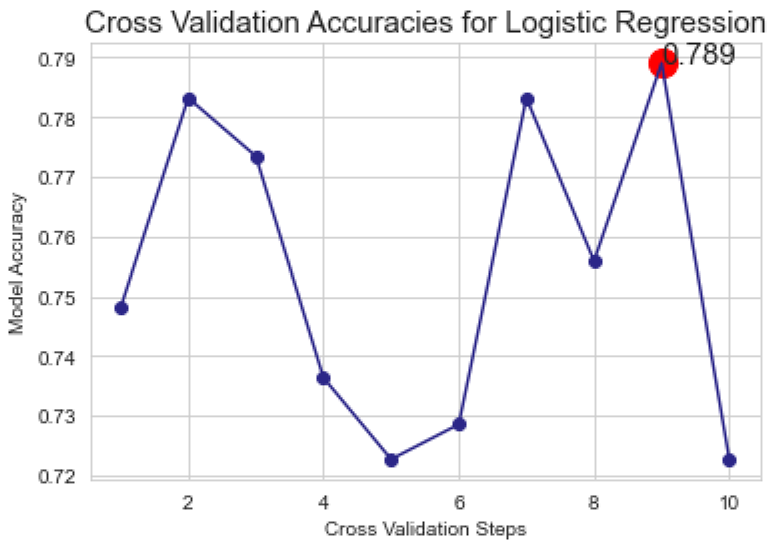
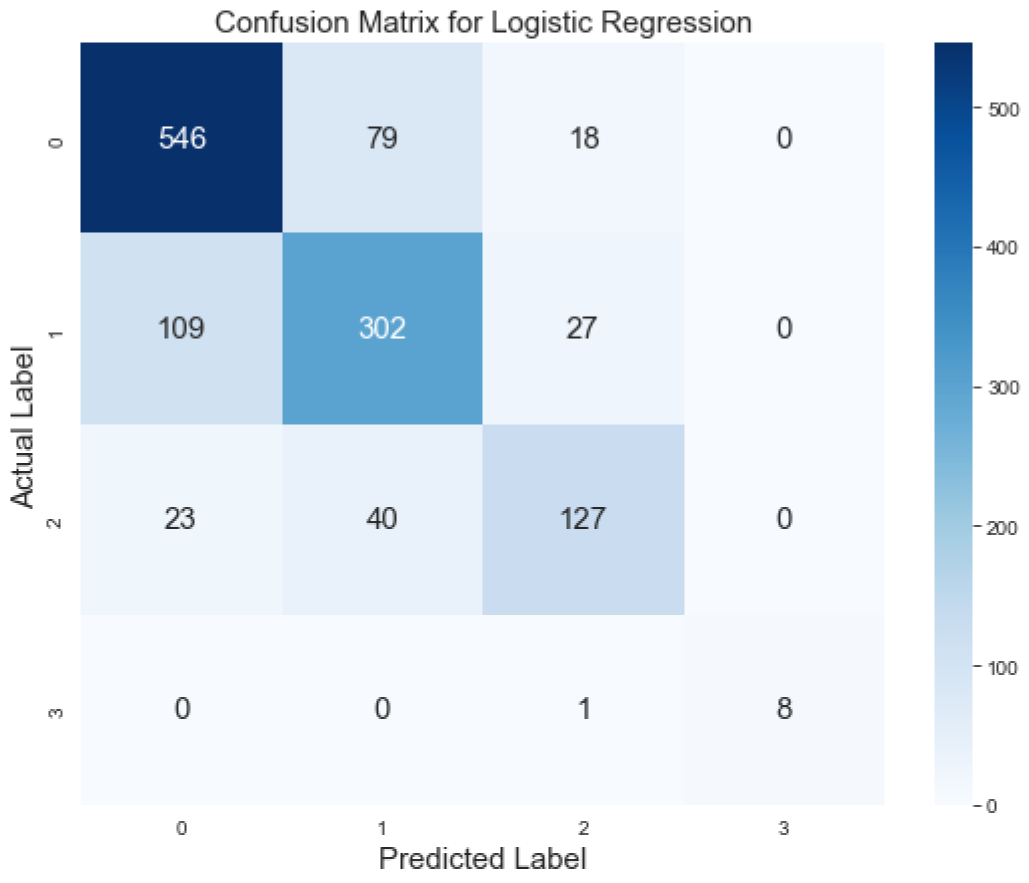


Fig 3.1.1 : Cross validation accuracies for Logistic Regression



Actual \ Predicted	0	1	2	3
0	546	79	18	0
1	109	302	27	0
2	23	40	127	0
3	0	0	1	8

Fig 3.1.2 : Confusion Matrix for Logistic Regression

3.2 Support Vector Machine
This method resulted in highest test accuracy of 98%, subjected to C = 10 and kernel = poly.

Classification Report for Test Data:

	Precision	Recall	F1 Score	Accuracy
Average	0.99	0.99	0.99	0.98

As expected, the overall accuracy for SVM was highest, given the nature of data.

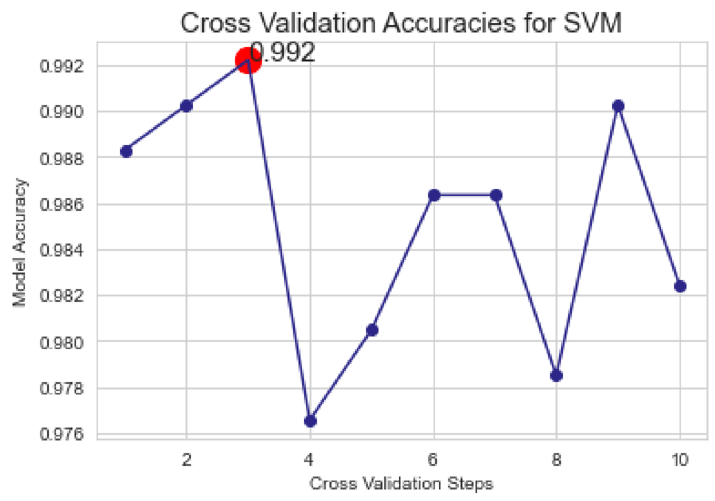


Fig 3.2.1 : Cross validation accuracies for SVM

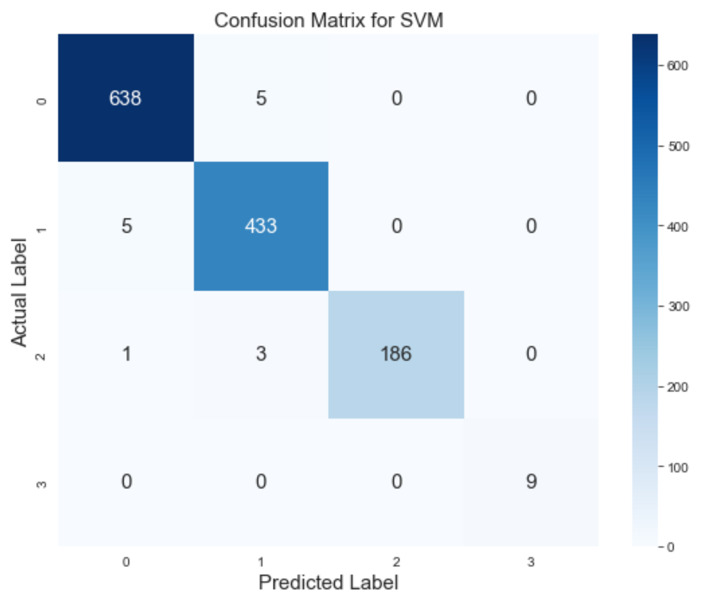


Fig 3.2.2 : Confusion Matrix for SVM

3.3 Random Forest

This method resulted in an optimal training accuracy of 73%, subjected to 'n_estimators': 1000 hyperparameters. The different cross validation accuracies can be summarized as follows:

Classification Report for Test Data:

	Precision	Recall	F1 Score	Accuracy
Average	0.64	0.42	0.42	0.72

As expected, the overall accuracy for this model was worst among all 3 models.

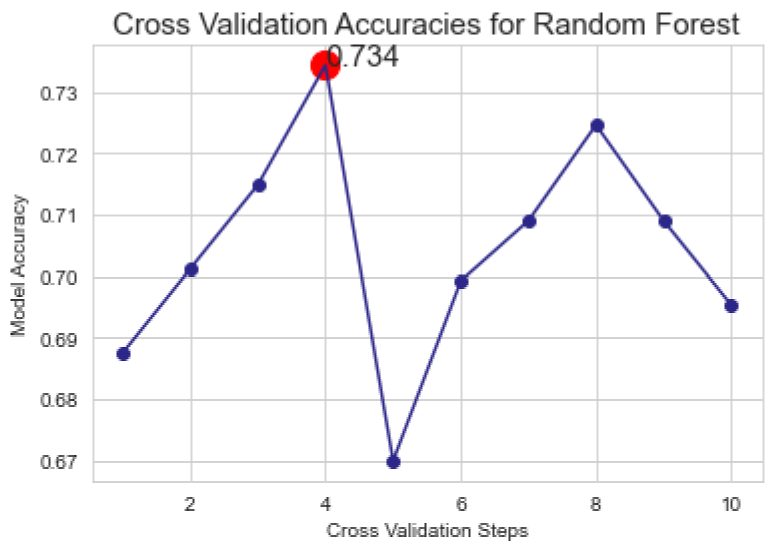


Fig 3.3.1 : Cross validation accuracies for Random Forest

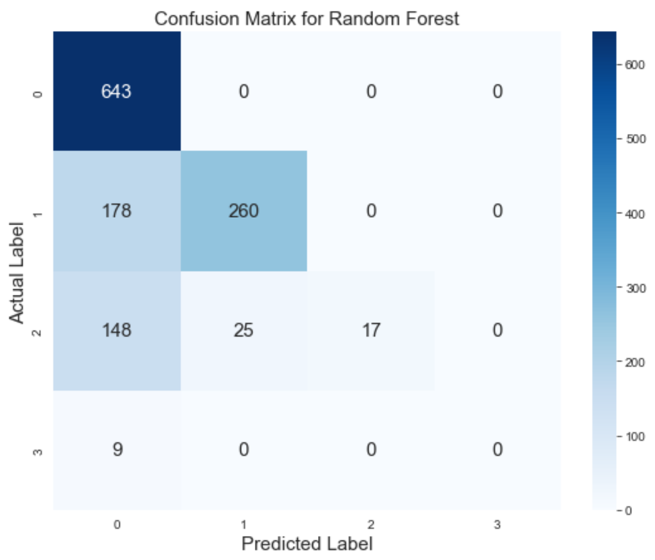


Fig 3.3.2 : Confusion Matrix for Random Forest

4. Conclusion

In this way we concluded a very successful data science project on predicting the severity of Alzheimer on the test dataset using MRI image as input. Initially we started with simpler Logistic Regression and Random Forest models, which weren't able to absorb the non-linear nature of data that well. Hence we finalized on an SVM model, for which hyperparameter tuning was done with corresponding KFold Cross Validation , resulting in a model with 98% accuracy in the test dataset.

5. References

1. "Alzheimer MRI Preprocessed Dataset." *Kaggle*, 2022, https://www.kaggle.com/datasets/sachinkumar413/alzheimer-mri-dataset. Accessed 26 November 2022.
2. "Alzheimer's disease - Symptoms and causes." *Mayo Clinic*, 19 February 2022, https://www.mayoclinic.org/diseases-conditions/alzheimers-disease/symptoms-causes/syc-20350447?utm_source=Google&utm_medium=abstract&utm_content=Alzheimers-disease&utm_campaign=Knowledge-panel. Accessed 26 November 2022.
3. Khattak, Wajid, et al. *Big Data Fundamentals: Concepts, Drivers & Techniques*. Edited by Thomas Erl, Prentice Hall, 2016.
4. "Support vector machine." *Wikipedia*, Wikipedia, 2022, https://en.wikipedia.org/wiki/Support_vector_machine. Accessed 26 November 2022.

6. Appendix

Attachment 1 : It runs the notebook for the entire data analysis.

```
#!/usr/bin/env python
# coding: utf-8

# In[499]:
import os
import pandas as pd
import numpy as np
from matplotlib import image

import seaborn as sns
import matplotlib.pyplot as plt

from skimage.transform import resize
from skimage.io import imread
from skimage.color import rgb2gray

from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split

from sklearn.pipeline import Pipeline
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report

sns.set_palette('Paired')
sns.set_style("whitegrid")
# In[386]:
class1 = 'Very_Mild_Demented/'
class2 = 'Mild_Demented/'
class3 = 'Moderate_Demented/'
class0 = 'Non_Demented/'
# ### Step 1 : Data Extraction / Cleaning / Aggregation
#
# 1. In this step, data is extracted from each source, cleaned, resized, flattened and finally
combined with all the sources to form an aggreagated dataframe that consists of all the classes.

# In[387]:
```

```
def file_append(class_path):
    image_array = []
    curr_path = os.path.join(os.getcwd(),class_path)
    cnt = 0
    file_list = [k for k in os.listdir(curr_path) if '.jpg' in k]
    for x in file_list:

        img_path = os.path.join(curr_path,x)

        img = imread(img_path)
#         img = rgb2gray(img)
        img = resize(img,(60,60,3))     ## Will need to change this resize parameter
        img = img.flatten()
        image_array.append(img)

    return image_array


class1_img = file_append(class1)
class2_img = file_append(class2)
class3_img = file_append(class3)
class0_img = file_append(class0)
# In[388]:
df1 = pd.DataFrame(class1_img)
df1['y']  = 1
df2 = pd.DataFrame(class2_img)
df2['y']  = 2
df3 = pd.DataFrame(class3_img)
df3['y']  = 3
df0 = pd.DataFrame(class0_img)
df0['y']  = 0

df = pd.concat([df0,df1,df2,df3],ignore_index=True)
df = df.sample(frac=1, random_state=42).reset_index(drop=True)
# ### Step 1.1 : Checking for null values

# In[389]:
df.isnull().sum().sum()
# ## Step 2 : EDA of variable

# In[390]:
sns.barplot(x =['Non-dementia','Very Mild', 'Mild', 'Moderate'],y = df['y'].value_counts())
plt.ylabel('Count')
plt.title('Class Wise Distribution of data')
# This distribution makes sense, as the percentage of people who have severe cases is much
lower than mild cases.
#
```

```
# ## Step 3 : Splitting entire dataset into train and test sets

# In[391]:
X,y = df[df.columns[:-1]],df['y']
X_train,X_test,y_train,y_test = train_test_split(X,y,train_size=0.8,random_state=121)
# ## Step 4 : Data Preprocessing pipeline for scaling and dimensionality reduction

# In[392]:
data_pipe = Pipeline([('scaler', StandardScaler()), ('pca',PCA(n_components=0.9))])
data_pipe.fit(X_train)
# In[393]:
X_train = data_pipe.transform(X_train)
X_test = data_pipe.transform(X_test)

y_train
# In[394]:
# pca = PCA(n_components=0.90)
# pca_data = pca.fit_transform(X_train)
# exp_var = pca.explained_variance_ratio_

print(f"Original Data dimension: {df.shape[1]} features")
print(f"After PCA Data dimension: {X_train.shape[1]} features")
# In[395]:
top10_var = [x*100 for x in data_pipe.named_steps['pca'].explained_variance_ratio_[:10]]
PC_labels = ['PC_'+str(x+1) for x in range(len(top10_var)) ]

plt.figure(figsize=(8,6))
plt.title("Variance Percentage for top 10 Principal Components")

bar = sns.barplot(x = top10_var,y = PC_labels)
for p in bar.patches:
    x = p.get_width()
    y = p.get_y()
    bar.annotate(format(x,'.1f'),(x,y),ha = 'center', va = 'center',size=12,color='black',xytext = (0,
-20), textcoords = 'offset points',bbox = dict(boxstyle = 'square, pad = 0.2', lw = 0.8, ec =
'#6e6e6e'))
#     print(p.get_width())

plt.title("Variance Ratio for top 10 Principal Components")
# In[398]:
scatter_plots = X_train[:,:5]
data_cols = ['PC_'+str(x+1) for x in range(scatter_plots.shape[1]) ]
scatter_df = pd.DataFrame(scatter_plots,columns=data_cols)
scatter_df['y'] = y_train.reset_index(drop=True)

label_map = {0:'No Dementia',1:'Very Mild', 2: 'Mild',3:'Moderate-High'}
```

```
scatter_df['y'] = scatter_df['y'].map(label_map)

sns.pairplot(scatter_df,hue="y",palette="husl",)
# From the above plot, it is very clear that , the data is not separable in lower dimensions, as
we can see in scatter plots. Hence it is not easy to find a linearly separable boundary, thus it is
my assumption that we might need to opt for SVM or other methods for high accuracy.

# ## Step 5 : Data Modelling

# In[688]:
#Base function for models

def train_model(model,search_grid):
    cv = KFold(n_splits=10)
    grid_search = GridSearchCV(estimator=model, param_grid=search_grid, n_jobs=-1, cv=cv,
scoring='accuracy',error_score=0)
    grid_result = grid_search.fit(X_train,y_train)

    return grid_result

def scatter_plots(score_array,title):
    sns.set_palette('CMRmap')
    lr_scores = [max(score_array.cv_results_['split'+str(i)+'_test_score']) for i in range(0,10)]
    sns.lineplot(range(1,11),lr_scores)
    j = plt.scatter(range(1,11),lr_scores)

    plt.xlabel('Cross Validation Steps')
    plt.ylabel('Model Accuracy')
    plt.title(f'Cross Validation Accuracies for {title}',size=15)

    for i,j in enumerate(lr_scores):
        if j == max(lr_scores):
            plt.scatter(i+1,j,c='red',marker='o',s=200)
            plt.annotate(format(j,'.3f'),(i+1,j),size=15)

def cf_plot(score_array,title):
    plt.figure(figsize=(9,7))
    cf_matrix = confusion_matrix(y_test,y_pred=score_array.predict(X_test))
    sns.heatmap(cf_matrix/np.sum(cf_matrix,axis=0), annot=True,
           fmt='.2%', cmap='Blues',annot_kws={"size":15})
    plt.title(f'Confusion Matrix for {title}',size=15)
    print(classification_report(y_test,y_pred=score_array.predict(X_test)))
    # ### Step 5.1 Logistic Regression

# In[673]:
lr_model = LogisticRegression(max_iter=100)
```

```
solvers = ['newton-cg']
penalty = ['l2','l3']
c_values = [ 1.0, 0.1, 0.01,0.001]
lr_search_grid = dict(solver=solvers,penalty=penalty,C=c_values)
lr_grid_result = train_model(lr_model,lr_search_grid)
# # grid_result.cv_results_
# In[670]:
scatter_plots(lr_grid_result,'Logistic Regression')
# In[689]:
cf_plot(lr_grid_result,'Logistic Regression')
# ### Step 5.2 : SVM

# In[515]:
svm_model = SVC()
c_ = [0.5,1,2,5,10]
kernel = ['rbf','poly']

search_parameters = dict()
search_parameters['C'] = c_
search_parameters['kernel'] = kernel

svm_grid_result = train_model(svm_model,search_parameters)
svm_grid_result.best_params_
# In[684]:
svm_grid_result.score(X_test,y_test)
# In[685]:
svm_grid_result.best_params_
# In[682]:
scatter_plots(svm_grid_result,'SVM')
# In[681]:
cf_plot(svm_grid_result,'SVM')
# ### 5.3 Random Forest

# In[516]:

rf_model= RandomForestClassifier()
n_estimators = [10, 100, 500,1000]
max_features = ['sqrt', 'log2']

rf_grid = dict(n_estimators=n_estimators,max_features=max_features)
rf_grid_result  = train_model(rf_model,rf_grid)

rf_grid_result.best_params_
# In[675]:
scatter_plots(rf_grid_result,'Random Forest')
# In[680]:
cf_plot(rf_grid_result,'Random Forest')
```