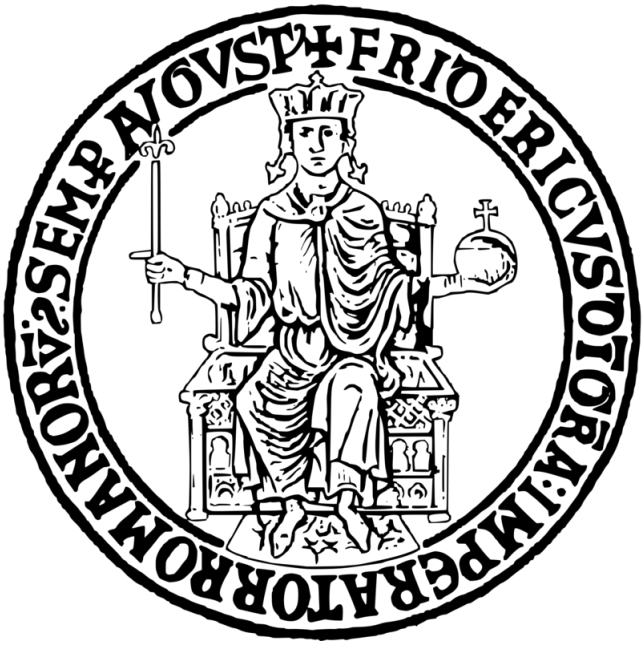
Models for Classification and Prediction of Alzheimer's Disease



MASTER'S DEGREE DATA SCIENCE
UNIVERSITY OF NAPLES FEDERICO II

Italo Alberto Ferrante P37000005
Francesco Romano P37000022

To my grandma, Ninella, patient at terminal stage of Alzheimer's Desease
and Black, my dog forever
Italo

ABSTRACT

There is no one test to determine if someone has dementia. Doctors diagnose Alzheimer's and other types of dementia based on a careful medical history, a physical examination, laboratory tests, and the characteristic changes in thinking, day-to-day function and behavior associated with each type. Doctors can determine that a person has dementia with a high level of certainty. But it's harder to determine the exact type of dementia because the symptoms and brain changes of different dementias can overlap. In some cases, a doctor may diagnose "dementia" and not specify a type. If this occurs it may be necessary to see a specialist such as a neurologist or gero-psychologist.

Contents

1 Introduction

During the past two decades there has been a substantial improvement in healthcare thanks to the implement of new technologies to aid researchers. However there are still more challenges to face against popular yet unknown diseases.
One of these battles is against *Alzheimer*, despite being discovered more than a hundred years ago, researchers still have few answers to what can cause, prevent or cure the disease. In 2015 there were almost 30 millions people worldwide suffering AD, these numbers hold a large weight on Earth society: the socioeconomic cost to aid those affected by AD is notable. Numbers vary between studies but dementia costs worldwide have been calculated around 160 billion dollars.
Alzheimer's disease is a form of neurodegenerative disease that can lead to dementia and death. The first symptoms are attributed to ageing or stress and can consist in short memory loss, late stages of the disease can lead to poor ability to think or speak. The cause of AD is still poorly understood, and it's substantially important for researchers to locate causes to help and prevent the patient from suffering the latest stages of the disease.
Our work focuses on prediction and recognition of patients suffering AD and it's divided in two sections:

- *Prediction Analysis*: We applied Machine Learning methods to a data-set of MRI subjects trying to analyze and predict dementia.
- *Deep Learning Analysis*: Lastly, using a data-set of MRI scans images, we trained a Convolutional Neural Network to recognize the disease in every stage of developing, from mild to moderate.

2 Preliminary work

2.1 Dataset Description

Dataset for Machine Learning Analysis is freely downloadable on Kaggle [1] and, as we can read on the context, dataset is made available by the Washington University Alzheimer's Disease Research Center, Dr. Randy Buckner at the Howard Hughes Medical Institute (HHMI)(at Harvard University, the Neuroinformatics Research Group (NRG) at Washington University School of Medicine, and the Biomedical Informatics Research Network (BIRN). This set consists of a cross-sectional collection of 416 subjects aged 18 to 96. For each subject, 3 or 4 individual T1-weighted MRI scans obtained in single scan sessions are included. The subjects are all right-handed and include both men and women. 100 of the included subjects over the age of 60 have been clinically diagnosed with very mild to moderate Alzheimer's disease (AD). Additionally, a reliability data set is included containing 20 nondemented subjects imaged on a subsequent visit within 90 days of their initial session. For this work , our inspiration was "Can we predict Dementia?" and, as preliminary part, we started by performing EDA.

2.2 Exploratory Data Analysis

In statistics, exploratory data analysis[2] is an approach to analyzing data sets to summarize their main characteristics, often with visual methods. A statistical model can be used or not, but primarily EDA is for seeing what the data can tell us beyond the formal modeling or hypothesis testing task. Exploratory data analysis was promoted by John Tukey to encourage statisticians to explore the data, and possibly formulate hypotheses that could lead to new data collection and experiments. EDA is different from initial data analysis (IDA), which focuses more narrowly on checking assumptions required for model fitting and hypothesis testing, and handling missing values and making transformations of variables as needed. EDA encompasses IDA. In particular, we were interested in Data Visualization that is[3] the graphic representation of data. It involves producing images that communicate relationships among the represented data to viewers of the images. This communication is achieved through the use of a systematic mapping between graphic marks and data values in the creation of the visualization. This mapping establishes how data values will be represented visually, determining how and to what extent a property of a graphic mark, such as size or color, will change to reflect changes in

[1] https://www.kaggle.com/jboysen/mri-and-alzheimers
[2] https://en.wikipedia.org/wiki/Exploratory_data_analysis
[3] https://en.wikipedia.org/wiki/Data_visualization

the value of a datum.
Working on Jupyter Notebook, we initially imported these libraries:

```
import numpy as np
from matplotlib import pyplot as plt
import pandas as pd
import seaborn as sns
```

Figure 1: EDA libraries

NumPy[4] aims to provide an array object that is up to 50x faster that traditional Python lists. Matplotlib.pyplot[5] is a collection of functions that make matplotlib work like MATLAB. Each pyplot function makes some change to a figure: e.g., creates a figure, creates a plotting area in a figure, plots some lines in a plotting area, decorates the plot with labels, etc. pandas[6] is a fast, powerful, flexible and easy to use open source data analysis and manipulation tool, built on top of the Python programming language, whilst Seaborn[7] is a Python data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics.
Many records of the dataset had a lot of missing values, so we decided to remove them because they were totally useless. Shape, after this operation, dropped to 216 statistical units and 12 features.
Below is shown a view of the dataset before removals and after taking away 'ID', 'Hand' and 'Delay' columns because useless (in particular every individual in the dataset was right-handed, so this data had no value):

	ID	M/F	Hand	Age	Educ	SES	MMSE	CDR	eTIV	nWBV	ASF	Delay
0	OAS1_0001_MR1	0	R	74	2	3	29	Nondementia	1344	0.743	1,306	NaN
1	OAS1_0002_MR1	0	R	55	4	1	29	Nondementia	1147	0.810	1,531	NaN
2	OAS1_0003_MR1	0	R	73	4	3	27	Very Mild Dementia	1454	0.708	1,207	NaN
3	OAS1_0010_MR1	1	R	74	5	2	30	Nondementia	1636	0.689	1,073	NaN
4	OAS1_0011_MR1	0	R	52	3	2	30	Nondementia	1321	0.827	1,329	NaN

Figure 2: Dataset overview before drops

[4]https://www.w3schools.com/python/numpy_intro.asp
[5]https://matplotlib.org/tutorials/introductory/pyplot.html
[6]https://pandas.pydata.org/
[7]https://seaborn.pydata.org/

	M/F	Age	Educ	SES	MMSE	CDR	eTIV	nWBV	ASF
0	0	74	2	3	29	Nondementia	1344	0.743	1,306
1	0	55	4	1	29	Nondementia	1147	0.810	1,531
2	0	73	4	3	27	Very Mild Dementia	1454	0.708	1,207
3	1	74	5	2	30	Nondementia	1636	0.689	1,073
4	0	52	3	2	30	Nondementia	1321	0.827	1,329

Figure 3: Dataset overview after drops

In table below are explained meanings of variables:

Feature	**Meaning**
M/F	Gender of individual
Age	Age of individual
Educ	Education Index. It varies from 2 to 5
SES	Socio-Economic Status
MMSE	Mini-Mental State Examination score (range is from 0 = worst to 30 = best)
CDR	Clinical Dementia Rating (0 = no dementia, 1 = mild AD, 2 = moderate AD, 3 = very mild AD)
eTIV	Estimated total intracranial volume, mm3
nWBV	normalized Whole Brain Volume, expressed as a percent of all voxels in the atlas-masked image that are labeled as gray or white matter by the automated tissue segmentation process
ASF	defined as the volume-scaling factor required to match each individual to the atlas target. Atlas normalization, as commonly used by functional data analysis, provides an automated solution to the widely encountered problem of correcting for head size variation in regional and whole-brain morphometric analyses, so long as an age- and population-appropriate target atlas is used

Table 1: Features Explaination

In particular, as we can see by this histogram, modalities of our reference feature (CDR) are not identically distributed:

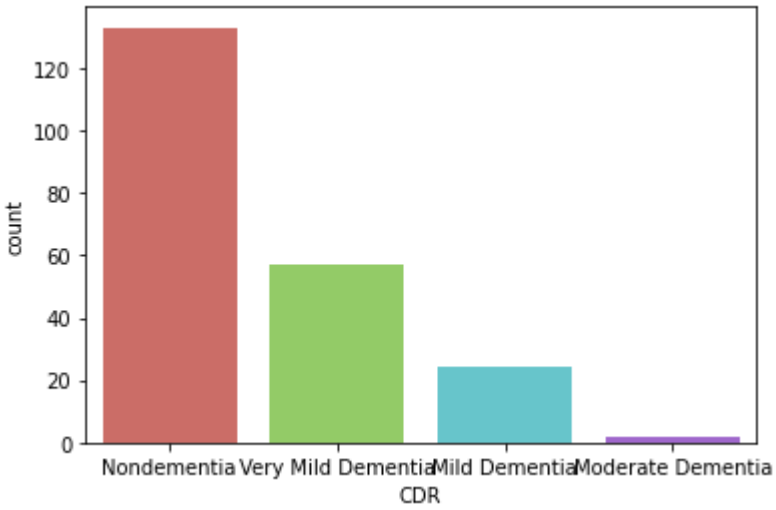


Figure 4: Frequencies of CDR modalities

This is a clear indication that dataset is unbalanced.
Distribution of ages is well designed by this chart:

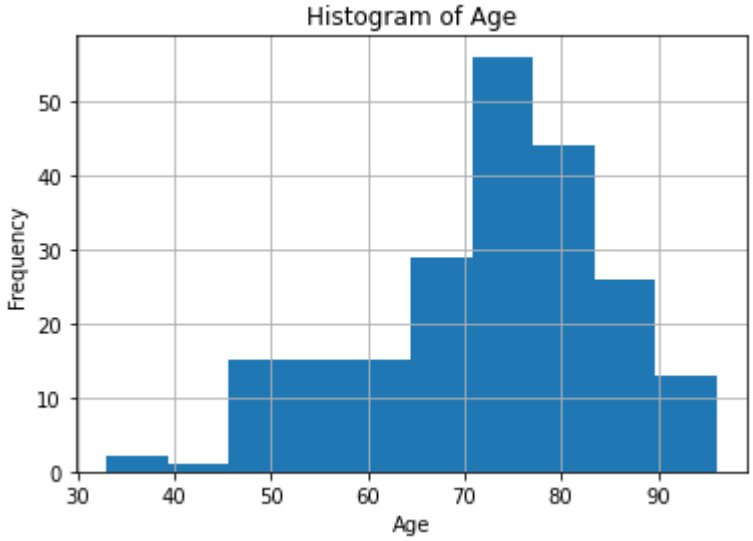


Figure 5: Distribution of age

Let's see means of non categorical variables per CDR status:

CDR	Age	MMSE	eTIV	nWBV
Mild Dementia	78.458333	21.958333	1477.791667	0.705500
Moderate Dementia	82.000000	15.000000	1456.500000	0.684000
Nondementia	69.233083	29.097744	1439.443609	0.769188
Very Mild Dementia	77.070175	25.877193	1495.438596	0.728175

Figure 6: Means of non categorical variables per CDR

For categorical variables, we preferred observe medians per CDR status.

CDR	M/F	Educ	SES
Mild Dementia	0.0	2.0	3.0
Moderate Dementia	0.5	2.0	3.5
Nondementia	0.0	4.0	2.0
Very Mild Dementia	0.0	3.0	3.0

Figure 7: Medians of categorical variables per CDR

As we can see, most of individuals with Nondementia, Very Mild Dementia and Dementia are men, while with Moderate Dementia there is an equal number of men and women. Much more interesting is to note how with the increasing of Education Index CDR status is lower. Stacked Bar below gives us confirmation:

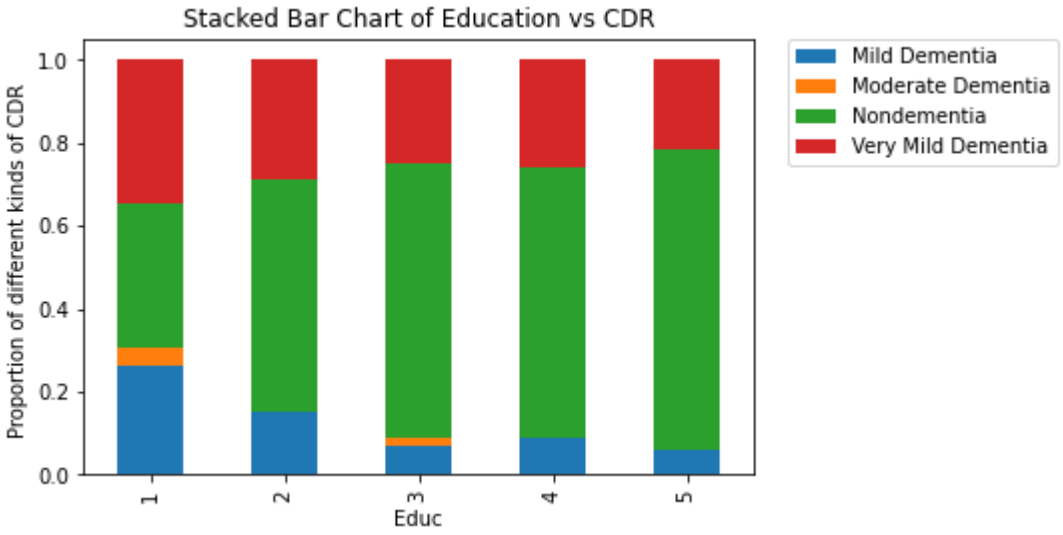


Figure 8: Stacked Bar of Education Status per CDR status

On the contrary, with the increasing of Socio-Economic Status, CDR is higher. Graphically, we plotted this crossbar:

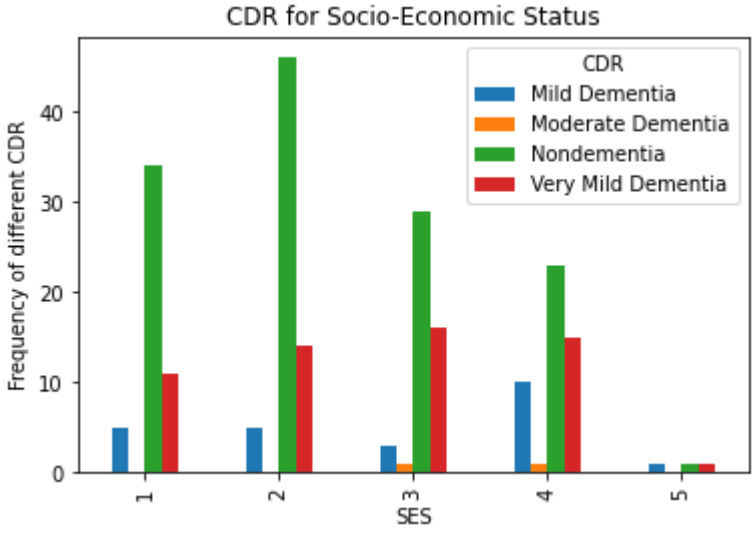


Figure 9: Crossbar for SES per CDR status

To verify intuitive concept that Dementia is a disease of the elderly, we considered scatterplots of classes.
Let's start with Nondementia patiens:

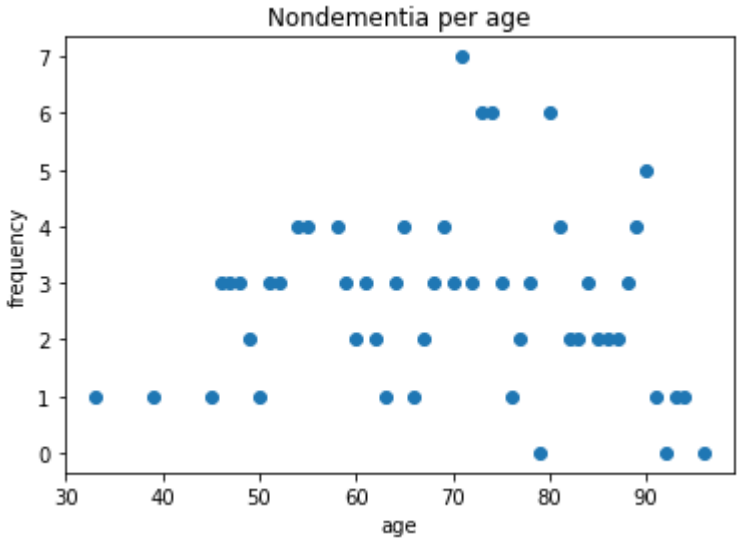


Figure 10: Scatterplot with individuals with Nondementia

This chart does not give us much information, specially if we consider that this is the most represented class. So, let's consider the second class, Very Mild Dementia:

Figure 11: Scatterplot with individuals with Very Mild Dementia

Chart clearly shows that patients with Very Mild Dementia are in a range from, roughly, 70 to 90 years.

We can see that is the same for individuals with Mild Dementia:

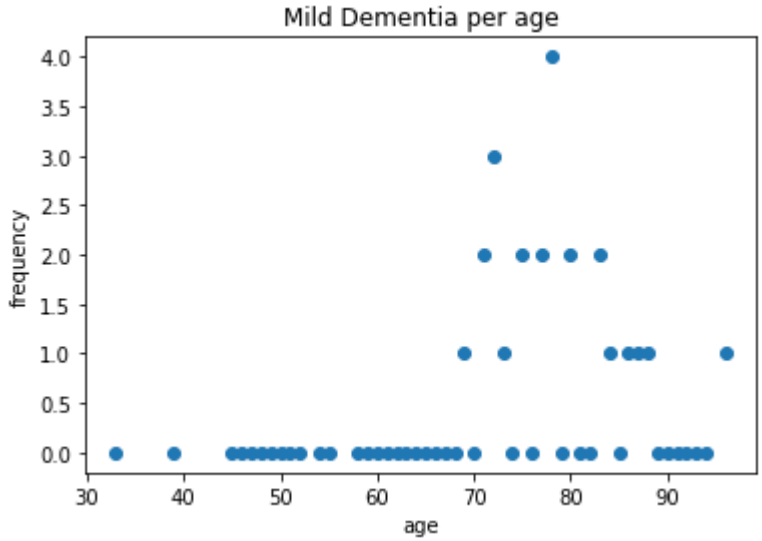


Figure 12: Scatterplot with indivuals with Mild Dementia

The two patients with Moderate Dementia are collected in this scatterplot:

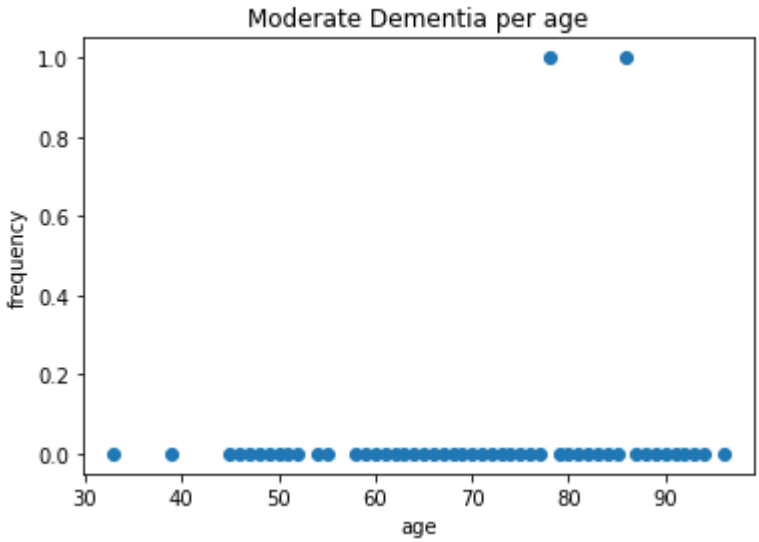


Figure 13: Scatterplot with individuals with Moderate Dementia

3 Logistic Regression

3.1 Logistic Regression with SMOTE

Oversampling Data As mentioned in the last section, our data is quite unbalanced, we have almost sixty Very Mild Dementia observations but only two Moderate Dementia cases. The challenge of working with imbalanced datasets is that most machine learning techniques will ignore, and in turn have poor performance on, the minority class, although typically it is performance on the minority class that is most important.
One approach to addressing imbalanced datasets is to oversample the minority class. The simplest approach involves duplicating examples in the minority class, although these examples don't add any new information to the model. Instead, new examples can be synthesized from the existing examples. This is a type of data augmentation for the minority class and is referred to as the **Synthetic Minority Oversampling Technique**, or **SMOTE** for short.
In order to oversample our data we imported the following libraries:

```
from imblearn.over_sampling import SMOTE
from sklearn import preprocessing
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
```

Figure 14: SMOTE libraries

During this step we split our dataset into test and train, for both our predictive and target variables, with a 70/30 ratio. Then we apply the oversampling function to the data, this is the result:

Previously we had 216 observations which only two of them of 'Moderate Dementia', now we have 364 observations and the same amount for each outcome of our target variable. We can start working towards our Regression models with a much more balanced dataset that will lead to better results.

Multilinear Logistic Regression In statistics, the *logistic model*[8] is used to model the probability of a certain class or event existing such as pass/fail, win/lose, alive/dead or healthy/sick. It's used to model a binary dependent variable, although many

[8] https://en.wikipedia.org/wiki/Logistic_regression

```
length of oversampled data is  364
Number of Nondementia in oversampled data 91
Number of Very Mild Dementia 91
Number of Moderate Dementia 91
Number of Mild Dementia 91
Proportion of Nondementia in oversampled data is  0.25
Proportion of Very Mild Dementia in oversampled data is  0.25
Proportion of Moderate Dementia in oversampled data is  0.25
Proportion of Mild Dementia in oversampled data is  0.25
```

Figure 15: Oversampled data output

more complex extensions exist. Outputs with more than two values are modeled by *multinomial logistic regression*[9]. The multinomial model is used when the dependent variable in question is nominal (equivalently categorical, meaning that it falls into any one of a set of categories that cannot be ordered in any meaningful way) and for which there are more than two categories. In our project case we have our target variable **CDR** consisting in four different categories (Non Dementia, Very Mild, Mild and Moderate Dementia). This is a *classification* problem and thus multinomial logistic regression fits best our purpose.
First thing first we import these libraries:

```
from sklearn.metrics import confusion_matrix, accuracy_score
from sklearn.linear_model import LogisticRegression
from sklearn import linear_model
import numpy as np
from sklearn import metrics
from sklearn.preprocessing import StandardScaler
```

Figure 16: Multinomial Logistic Regression libraries

Other than NumPy this time we import different sub-libraries of the same package: **sklearn**, also called scikit-learn[10]. The package is a set of simple and effective tools for predictive data analysis and other machine learning models, built on NumPy, SciPy and matplotlib.

With these packages we initially perform a feature scaling on our subsamples.

[9] https://en.wikipedia.org/wiki/Multinomial_logistic_regression
[10] https://scikit-learn.org/stable/index.html

After fitting the regression model into our training set, we predict the test set results using our fitted data. In the end our test score is **0.81**.

Confusion Matrix Despite having a good test score, in order to gather more insight on the model's result we plot our confusion matrix. A confusion matrix, is a specific table layout that allows visualization of the performance of an algorithm. Each row of the matrix represents the instances in a predicted class while each column represents the instances in an actual class (or vice versa).
This is the code for plotting the confusion matrix:

```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import metrics

cm = metrics.confusion_matrix(y_test, y_pred)
plt.figure(figsize=(9,9))
sns.heatmap(cm, annot=True, fmt=".3f", linewidths=.5, square = True, cmap = 'Blues_r');
plt.ylabel('Actual label');
plt.xlabel('Predicted label');
all_sample_title = 'Accuracy Test Score: {0}'.format(acc_score)
plt.title(all_sample_title, size = 15)
plt.show()
```

Figure 17: Python code for the Confusion Matrix

Seaborn is a Python data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics[11].

[11] https://seaborn.pydata.org

This is the output:

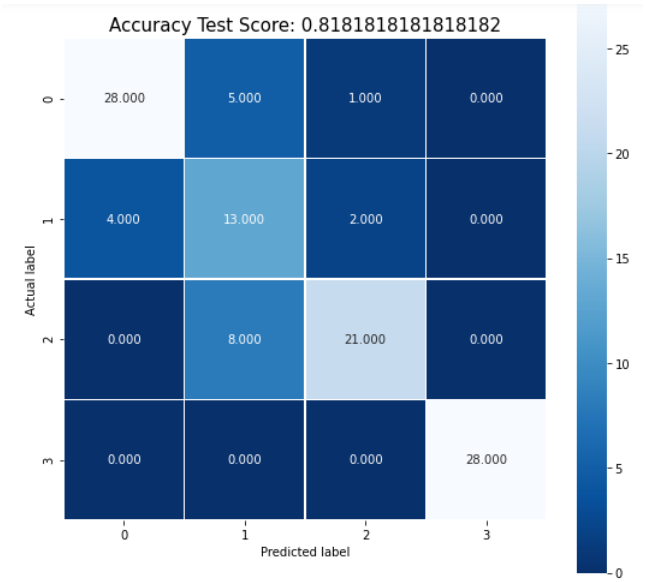


Figure 18: Confusion Matrix for Multinomial Logistic Regression post SMOTE

Looking at the matrix it seems like the model it's best performing on predicting 'Very Mild' and 'Moderate' dementia (respectively '2' and '3' as labels of the matrix). Instead the worst prediction comes from 'Mild' dementia outcomes.

Recursive Feature Elimination Because our dataset presents many variables, even after removing the useless ones such as "Hand" and "Delay", we tried to repeat the regression process after a feature selection. The method used for this step is **RFE**[12] or Recursive Feature Elimination. RFE is a feature selection method that fits a model and removes the weakest feature (or features) until the specified number of features is reached. In our case we wanted to select the four strongest features; We used the "RFE" branch of the *sklearn* library. The model shows a ranking of every variables (except the target one) ranging from three to one, where one is the strongest and three the weakest, in this case the strongest features are *Educ* (education), *SES* (social-economic status), *MMSE* (Mini-Mental State Examination score) and *nWBV* (normalized Whole Brain Volume). We then perform the Multinomial Logistic Regression using only these features and the test score is still **0.8**.
We decide to plot the confusion matrix for this step to look at any shift in performances:

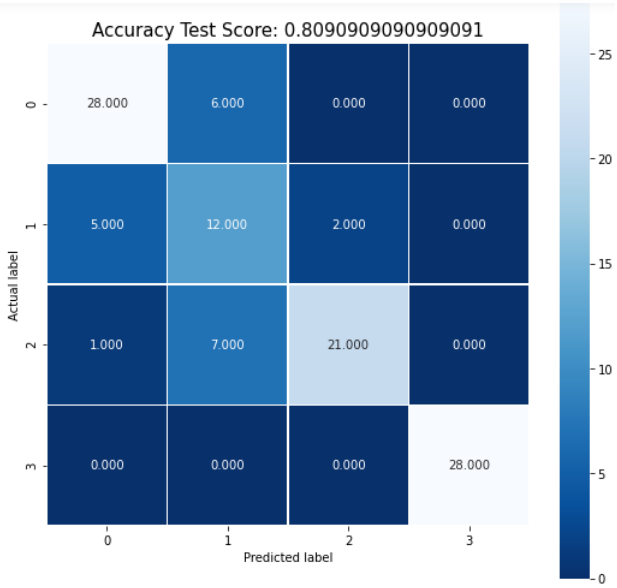


Figure 19: Confusion Matrix post RFE

[12] https://www.scikit-yb.org/en/latest/api/model_selection/rfecv.html

3.1.1 Merging Label 2 and 3 on df post SMOTE

To investigate further on our imbalanced dataset we tried a different approach. We decided to merge the last two labels (Mild Dementia and Moderate Dementia) to better balance our classes. So, we augmentated with SMOTE again (getting a proportion of each class of 0.33 this time) and re-implemented Logistic Regression, having an accuracy around **0.72**.

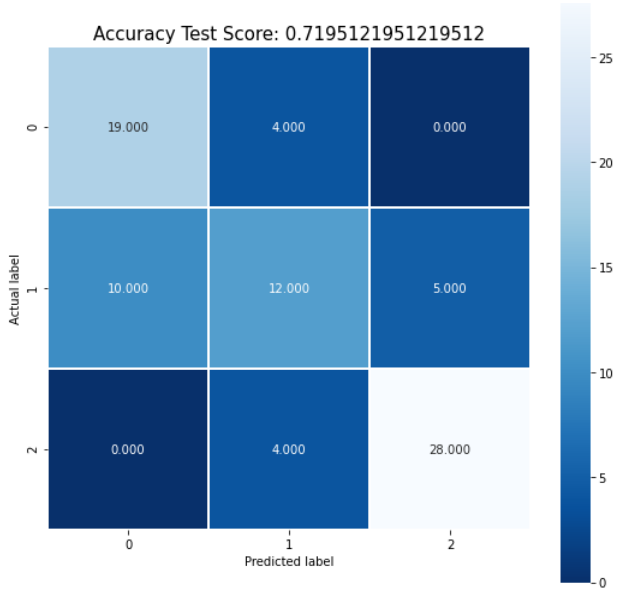


Figure 20: Confusion Matrix on df post RFE with label 2 and 3 merged

This time, results are not quite satisfactory, so we re-made RFE and applied resulting features ('M/F', 'MMSE', 'nWBV' and 'ASF')on new Logistic Regression, but results this time again do not convince to us.

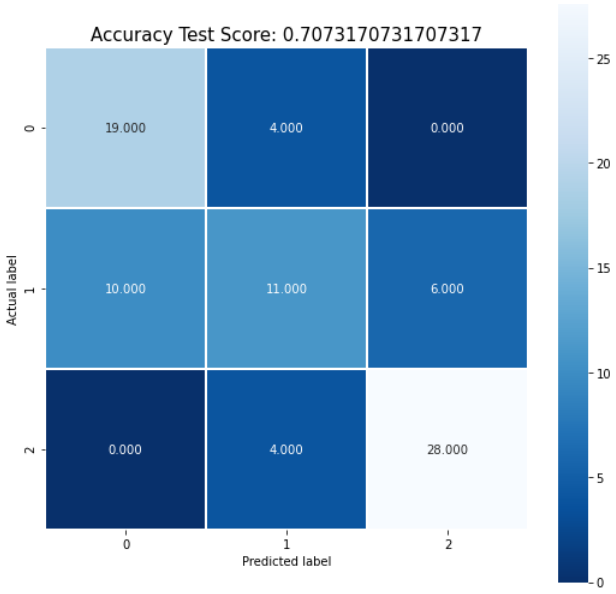


Figure 21: Confusion Matrix on df with label 2 and 3 merged post SMOTE post RFE

3.2 Logistic without SMOTE

To evaluate performance of starting dataframe we also performed Logistic Regression without SMOTE, expecting to have difficulties to predict Moderate Dementia because of low cardinality. Before, we preprocessed data like we did before and then divided data for training and testing. Note that for better perform a logistic regression with sklearn we standardized our data. Test Score achieved is **0.8153**. Results are displayed in confusion matrix below:

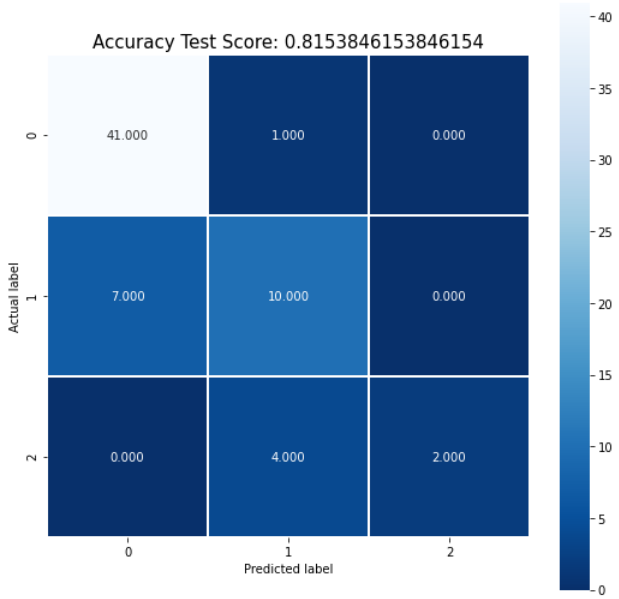


Figure 22: Confusion Matrix of starting dataframe

As, before we performed Recursive Feature Selection and chose first four variables, that were 'M/F', 'MMSE', 'nWBV' and 'ASF' this time. So, we re-run logreg and gained an accuracy equals to **0.81** (like before) and this Confusion Matrix:

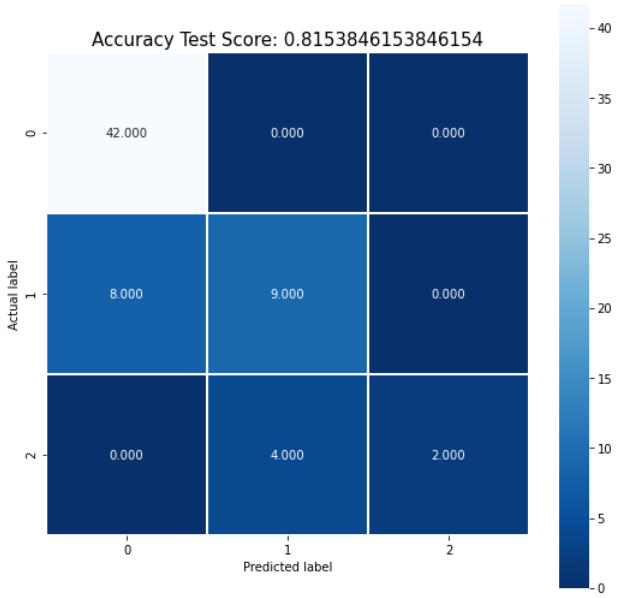


Figure 23: Confusion Matrix of starting dataframe post RFE

As we can see, models were not able to predict elements in class Moderate Dementia because of its low numerosity.

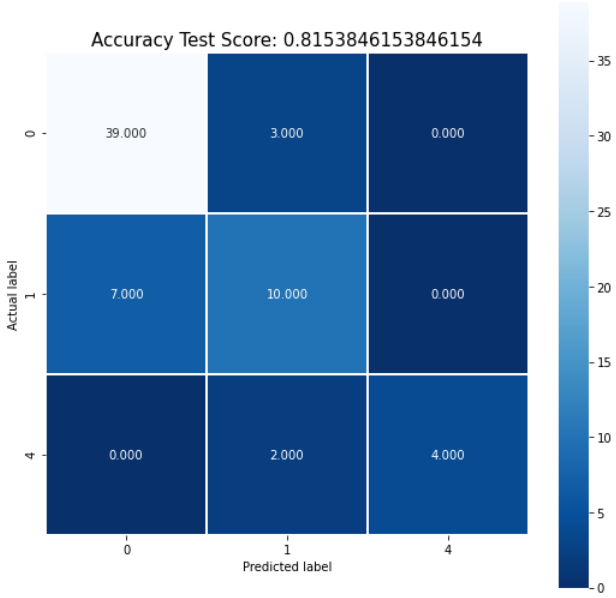


Figure 24: Confusion Matrix of starting dataframe

3.2.1 Merging label 2 and 3 on starting df

In an attempt to get better results we merged Mild Dementia and Moderate Dementia. We implemented again Logistic Regression and got an accuracy of **0.81**. Below is illustrated Confusion Matrix:

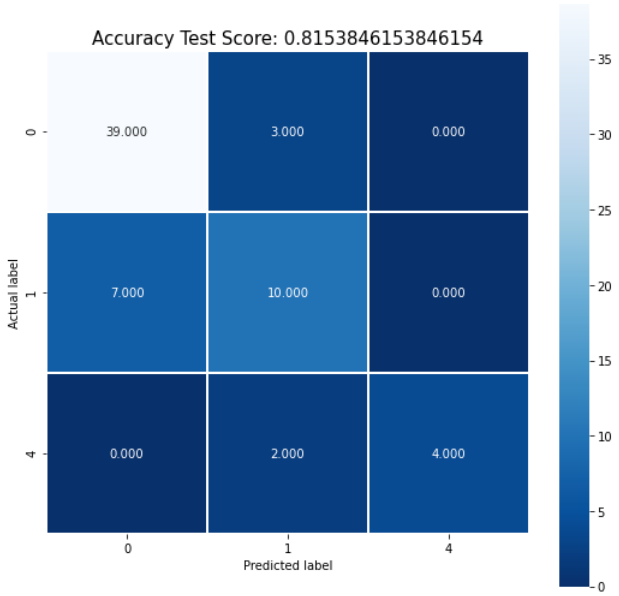


Figure 25: Confusion Matrix of starting dataframe with label 2 and 3 merged

As ever, we made RFE for the first four variables and run the algorithm of Logistic Regression with 'M/F', 'SES', 'MMSE' and 'nWBV'. In this way we have increased accuracy (climbed to **0.87**) and we can visualize this Confusion Matrix:

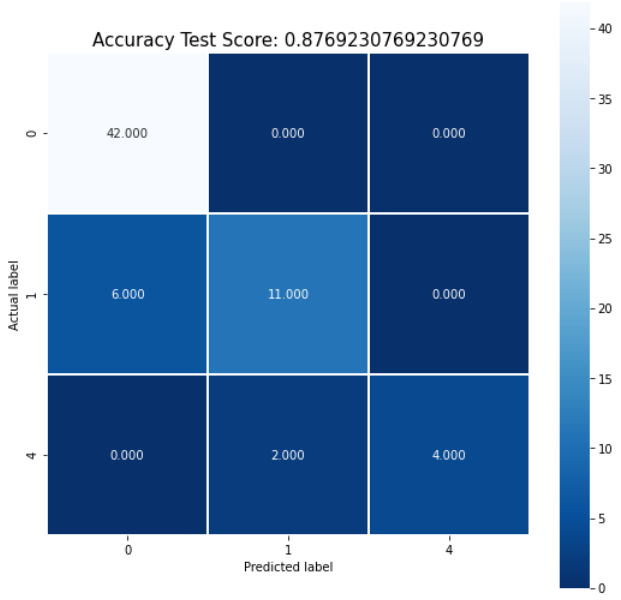


Figure 26: Confusion Matrix of starting dataframe with label 2 and 3 merged post RFE

This is a very impressive result, but we have to consider that this model does not take account of the difference between Mild Dementia and Moderate Dementia, so if we want to use this model should be necessary the implementation of a subclassifier able to discriminate between these two labels. This will be the object of part 7.

4 Decision Trees and Ensemble Methods

4.1 A brief overview

Decision Trees[13] are powerful algorithms that can perform regression tasks. One major problem with trees is their high variance. Often a small change in the data can result in a very different series of splits, making interpretation precarious. The major reason for this instability is the hierarchical nature of the process: the effect of an error in the top split is propagated down to all of the splits below it. Trees are also the fundamental component of *Random Forest*, one of the most powerful machine learning algorithm. Ensemble methods make predictions based on a number of different models to achieve higher flexibility. Two most popular ensemble methods are bagging and boosting. *Bagging* trains a bunch of individual models in a parallel way and is useful to reduce trees high variance. *Boosting* trains a models in a sequential way and each individual model learns from mistakes made by the previous one. It can combine several weak learners into a strong learner. Boosting is one of the most powerful learning idea introduced in the last twenty years.

Random Forest Random forest, one of the most powerful machine learning algorithm, is an ensemble model using bagging as the ensemble method and decision tree as the individual model. The Random Forest algorithm introduces extra randomness when growing trees; instead of searching for the very best feature when splitting a node, it searches for the best feature among a random subset of features. The algorithm results in greater tree diversity, which (again) trades a higher bias for a lower variance, generally yielding an overall better model.

AdaBoost In *AdaBoost* (Adaptive Boosting) predictors learn from the previous made mistakes. In particular, the AdaBoost algorithm involves using very short (one-level) decision trees, *stumps*, as weak learners that are added sequentially to the ensemble. Each subsequent model attempts to correct the predictions made by the model before it in the sequence. This is achieved by weighing the training dataset to put more focus on training examples on which prior models made prediction errors.

Gradient Boosting and XGBoost *Gradient Boosting*, just like AdaBoost, works by sequentially adding predictors to an ensemble, each one correcting its predecessor. This method tries to fit the new predictor to the residual errors made by the previous one and makes a new prediction by simply adding up the predictions (of

[13] Football Analysis - I. A. Ferrante, C. Santagata, A. Nappa, A. Laudante

all trees). *XGBoost* (Extreme Gradient Boosting) is an improvement of Gradient Boosting and aims to be extremely fast, scalable, and portable. XGBoost provides a parallel tree boosting (also known as GBDT, GBM) that solves many data science problems in a fast and accurate way. The same code runs on major distributed environments (Hadoop, SGE, MPI) and can solve problems beyond billions of examples.

4.2 Classification Trees and Ensemble Models

Tree Methods are very powerful but one of their weaknesses is that they need lots of data to perform well, and, unfortunately, our dataframe was not very wide-ranging, but we tried anyway to implement a decision tree.
We used sklearn library and, in particular, we imported DecisionTreeClassifier[14]
Using all features and running before a 10-fold Cross Validation, this is Classification Tree that we obtained:

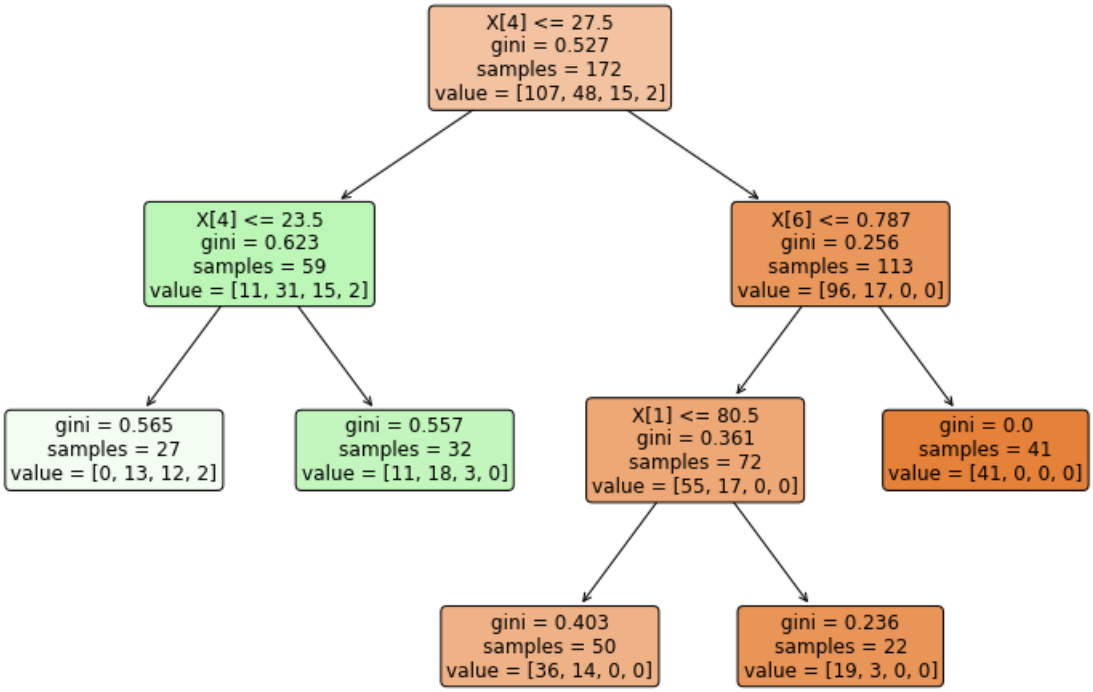


Figure 27: Decision Tree of starting df with every feature considered

Scores are written in table:

[14] https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html

Score	Value
MSE train	0.30
R^2 train	0.40
MSE	0.39
RMSE	0.62
MAE	0.34
MAPE	nan
Accuracy	nan
R^2	0.40

Table 2: Decision Trees Scores on starting dataframe

To balance the scarcity of records, we used only the first three most important features according to the previous RFE, that are M/F, MMSE and nWBV, and run again the algorithm (once again resampling with a 10-fold Cross Validation):

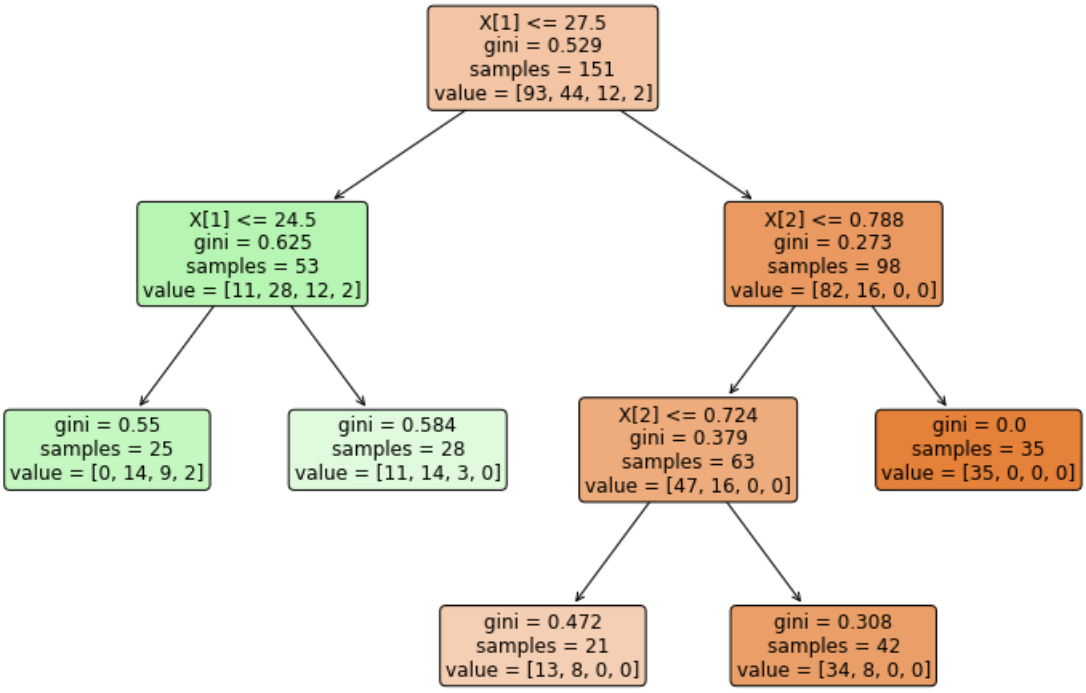


Figure 28: Decision Trees Scores on df with only M/F, MMSE and nWBV columns

These are scores of this model:

Score	**Value**
MSE train	0.31
R^2 train	0.36
MSE	0.32
RMSE	0.57
MAE	0.29
MAPE	nan
Accuracy	nan
R^2	0.47

Table 3: Decision Trees Scores on dataframe with only M/F, MMSE and nWBV columns

Results are not very satisfactory, so we decided to use dataframe post SMOTE because there are more records in the attempt to gain a stronger model. Resempling with 10-fold Cross Validation and using every features of dataframe post SMOTE we gained this Decision Tree:

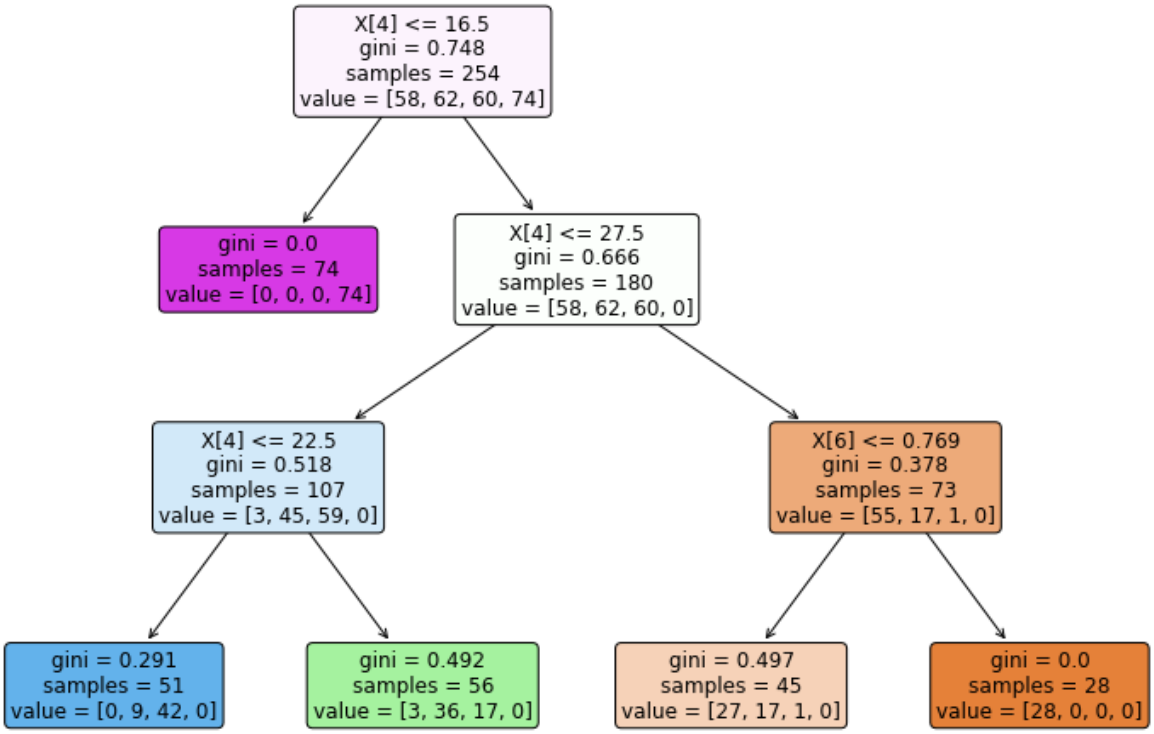


Figure 29: Decision Tree of df post SMOTE with every feature considered

In the table below are reported score of this classification model:

Score	**Value**
MSE train	0.2
R^2 train	0.84
MSE	0.25
RMSE	0.50
MAE	0.25
MAPE	nan
Accuracy	nan
R^2	0.77

Table 4: Decision Trees Scores on dataframe post SMOTE

We also performed Ensemble Methods.

For this project we worked with sklearn.ensemble[15] library, from which we imported BaggingClassifier, RandomForestClassifier, AdaBoostClassifier, GradientBoostingClassifier, XGBClassifier.
The goal of ensemble methods is to combine the predictions of several base estimators built with a given learning algorithm in order to improve generalizability / robustness over a single estimator. Two families of ensemble methods are usually distinguished:

- In averaging methods, the driving principle is to build several estimators independently and then to average their predictions. On average, the combined estimator is usually better than any of the single base estimator because its variance is reduced.
  Examples: Bagging methods, Forests of randomized trees, ...

- By contrast, in boosting methods, base estimators are built sequentially and one tries to reduce the bias of the combined estimator. The motivation is to combine several weak models to produce a powerful ensemble.
  Examples: AdaBoost, Gradient Tree Boosting, ...

Scores of ensemble methods and classification tree of dataframe post SMOTE are reported in the table below:

[15] https://scikit-learn.org/stable/modules/ensemble.html

	SCORES					
	Classification Tree	**Bagging**	**Random forest**	**AdaBoost**	**Gradient Boosting**	**XGBoost**
MSE	0.25	0.28	0.32	0.32	0.32	0.32
RMSE	0.50	0.53	0.57	0.57	0.57	0.57
MAE	0.25	0.28	0.29	0.29	0.32	0.32
MAPE	nan	inf	50.00	inf	inf	inf
Accuracy(%)	nan	inf	50.00	inf	inf	inf
R^2	0.77	0.55	0.47	0.47	0.47	0.47

Table 5: Scores of Ensemble Models and Classification Tree on df post SMOTE

5 Support Vector Machines

5.1 SVM with SMOTE

In machine learning, support-vector machines[16] are supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis. Given a set of training examples, each marked as belonging to one or the other of two categories, an SVM training algorithm builds a model that assigns new examples to one category or the other, making it a non-probabilistic binary linear classifier. An SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible. New examples are then mapped into that same space and predicted to belong to a category based on the side of the gap on which they fall. In addition to performing linear classification, SVMs can efficiently perform a non-linear classification using what is called the kernel trick, implicitly mapping their inputs into high-dimensional feature spaces. The kernel function that we used are:

- linear;

- polynomial;

- gaussian (rbf);

- sigmoid.

In harmony with work done so far, we implemented SVM algorithms on dataset post and before SMOTE to be able to make more effective comparison. Novelty introduced in this model is One-Hot Encoding. This technique consists in create as many new variables as are labels of categorical variables and map presence/absence of the modality with 1/0. We implemented One-Hot Encoding on X variables and got dataframe displayed in this view:

[16] https://en.wikipedia.org/wiki/Support$_vector_machine$

	Age	MMSE	eTIV	nWBV	ASF	M/F_0	M/F_1	Educ_1	Educ_2	Educ_3	Educ_4	Educ_5	SES_1	SES_2	SES_3	SES_4	SES_5
0	70	30	1660	0.739	1.057	0	1	0	0	0	0	1	1	0	0	0	0
1	88	28	1370	0.765	1.281	1	0	0	1	0	0	0	0	0	1	0	0
2	78	23	1462	0.697	1.200	1	0	1	0	0	0	0	0	0	0	0	1
3	90	25	1301	0.645	1.349	0	1	0	1	0	0	0	0	0	1	0	0
4	80	20	1494	0.665	1.175	0	1	0	1	0	0	0	0	0	0	1	0

Figure 30: X variables after One-Hot Encoding on categorical variables

At that point we proceeded to make a Simple (Linear) SVM. Below are reported metrics and relative Confusion Matrix:

	precision	recall	f1-score	support
0	0.81	0.76	0.78	33
1	0.56	0.54	0.55	26
2	0.75	0.90	0.82	20
3	1.00	0.97	0.98	31
accuracy			0.79	110
macro avg	0.78	0.79	0.78	110
weighted avg	0.79	0.79	0.79	110

Figure 31: Metrics of SVM Linear Model on df post SMOTE

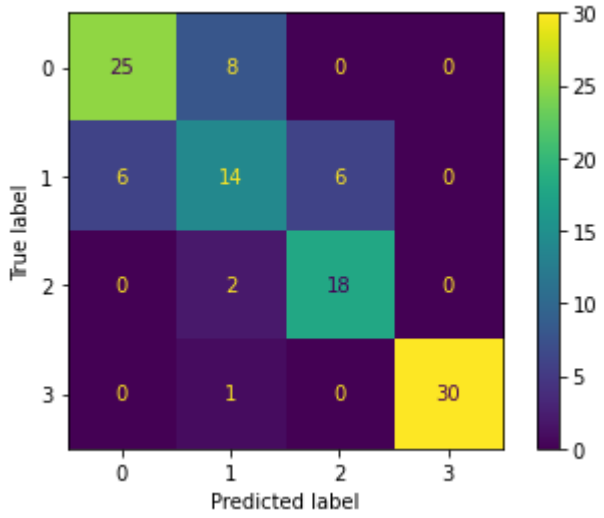


Figure 32: Confusion Matrix of SVM Linear Model on df post SMOTE

Model is very good, but we, however, tried to change kernel.
In order, metrics and Confusion Matrices with polynomial, gaussian and sigmoid kernels.

	precision	recall	f1-score	support
0	0.83	0.30	0.44	33
1	0.62	0.31	0.41	26
2	0.69	0.55	0.61	20
3	0.45	1.00	0.62	31
accuracy			0.55	110
macro avg	0.65	0.54	0.52	110
weighted avg	0.65	0.55	0.52	110

Figure 33: Metrics of SVM Polynomial Model on df post SMOTE

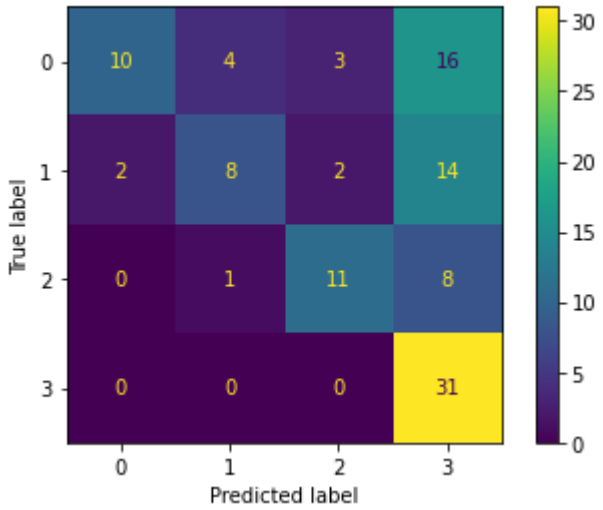


Figure 34: Confusion Matrix of SVM Polynomial Model on df post SMOTE

	precision	recall	f1-score	support
0	0.71	0.61	0.66	33
1	0.59	0.50	0.54	26
2	0.63	0.95	0.76	20
3	1.00	0.97	0.98	31
accuracy			0.75	110
macro avg	0.73	0.76	0.74	110
weighted avg	0.75	0.75	0.74	110

Figure 35: Metrics of SVM Gaussian Model on df post SMOTE

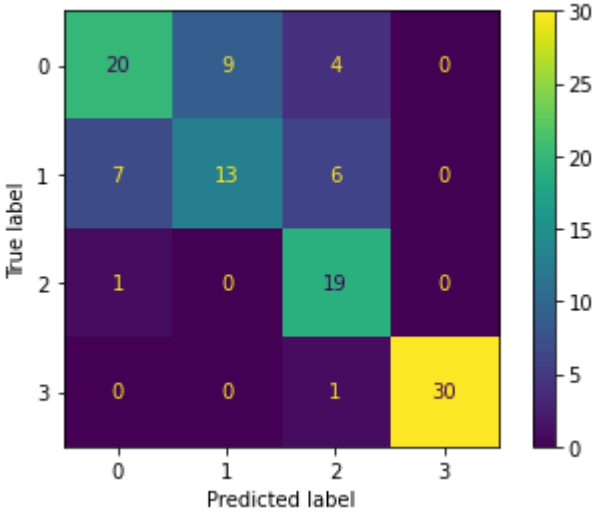


Figure 36: Confusion Matrix of SVM Gaussian Model on df post SMOTE

	precision	recall	f1-score	support
0	0.76	0.67	0.71	33
1	0.48	0.50	0.49	26
2	0.59	0.65	0.62	20
3	0.94	0.97	0.95	31
accuracy			0.71	110
macro avg	0.69	0.70	0.69	110
weighted avg	0.71	0.71	0.71	110

Figure 37: Metrics of SVM Sigmoid Model on df post SMOTE

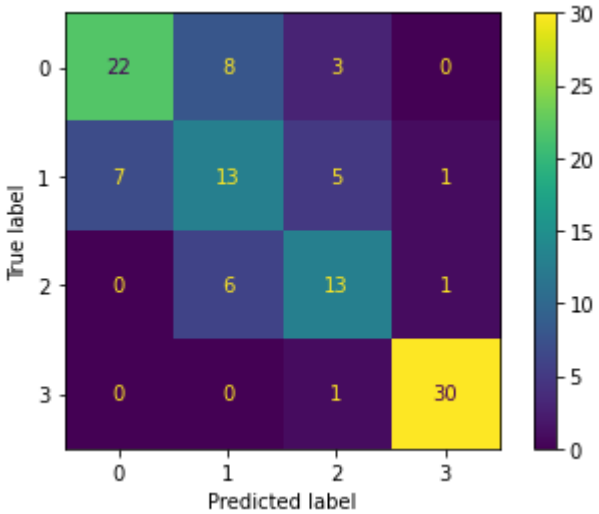


Figure 38: Confusion Matrix of SVM Sigmoid Model on df post SMOTE

5.1.1 Cross Validation on df post SMOTE

Except for Sigmoid Kernel, models are all very valid. To try to achieve better performances we made Cross Validation to optimize parameters. Parameters derived with a 5-fold CV are:

```
{'C': 10, 'gamma': 'scale', 'kernel': 'rbf'}
```

Figure 39: Parameters derived with 5-fold Cross Validation

So, we made re-run algorithm and with parameters we used kernel equals to rbf, 'C' equals to 10 and 'gamma' equals to 'scale'. The parameter C, common to all SVM kernels, trades off misclassification of training examples against simplicity of the decision surface. A low C makes the decision surface smooth, while a high C aims at classifying all training examples correctly. Gamma defines how much influence a single training example has. The larger gamma is, the closer other examples must be to be affected.[17]

[17] https://scikit-learn.org/stable/modules/svm.html

	precision	recall	f1-score	support
0	0.74	0.70	0.72	33
1	0.65	0.50	0.57	26
2	0.62	0.90	0.73	20
3	1.00	0.97	0.98	31
accuracy			0.76	110
macro avg	0.75	0.77	0.75	110
weighted avg	0.77	0.76	0.76	110

Figure 40: Metrics of SVM Gaussian Model on df post SMOTE with C = 10 and gamma = scale

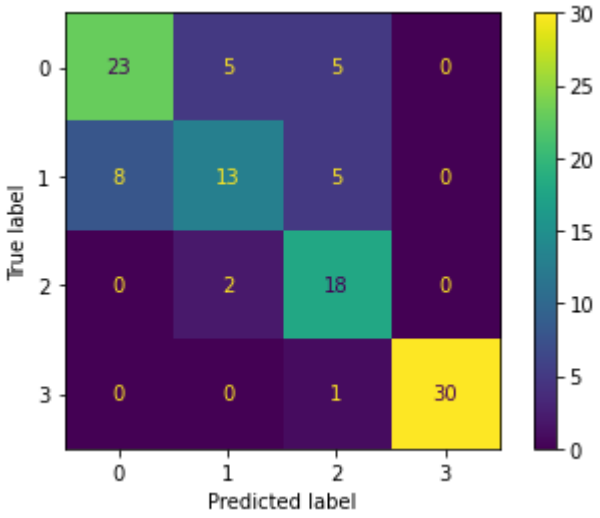


Figure 41: Confusion Matrix of SVM Gaussian Model on df post SMOTE with C = 10 and gamma = scale

5.1.2 Merging Mild Dementia and Moderate Dementia on df post SMOTE

To try to improve results, we merged label 'Very Mild Dementia' and 'Moderate Dementia'. Along the lines of before, we made SMOTE on dataframe with merged labels and got a new dataframe with lenght of 273 and a proportion of 0.33 for each label. From here, metrics and Confusion Matrices for different kernels (linear, polynomial, gaussian and sigmoid).

	precision	recall	f1-score	support
0	0.66	0.91	0.76	23
1	0.54	0.52	0.53	25
4	0.92	0.71	0.80	34
accuracy			0.71	82
macro avg	0.71	0.71	0.70	82
weighted avg	0.73	0.71	0.71	82

Figure 42: Metrics of SVM Linear Model on df with label 2 and 3 merged post SMOTE

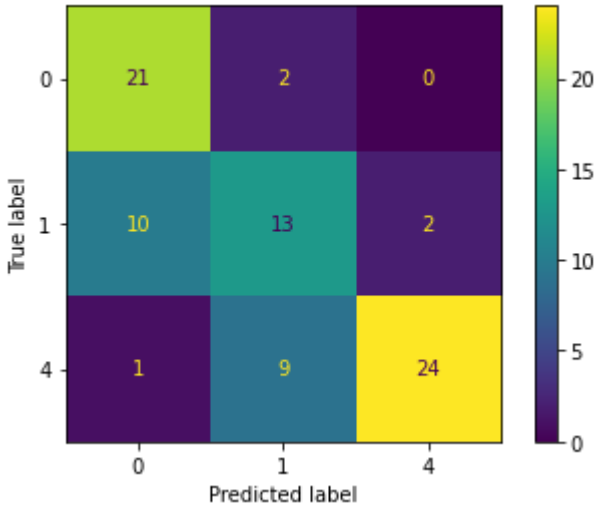


Figure 43: Confusion Matrix of SVM Linear Model on df with label 2 and 3 post SMOTE

	precision	recall	f1-score	support
0	0.70	0.30	0.42	23
1	0.46	0.84	0.59	25
4	0.92	0.71	0.80	34
accuracy			0.63	82
macro avg	0.69	0.62	0.61	82
weighted avg	0.72	0.63	0.63	82

Figure 44: Metrics of SVM Polynomial Model on df with label 2 and 3 merged post SMOTE

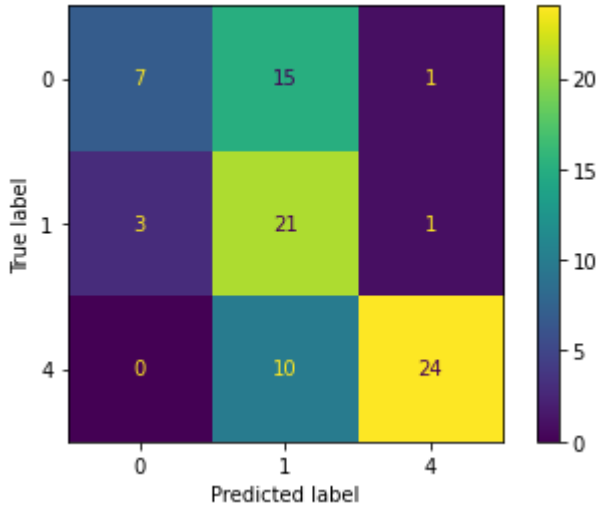


Figure 45: Confusion Matrix of SVM Polynomial Model on df with label 2 and 3 post SMOTE

	precision	recall	f1-score	support
0	0.71	0.87	0.78	23
1	0.63	0.68	0.65	25
4	0.93	0.74	0.82	34
accuracy			0.76	82
macro avg	0.76	0.76	0.75	82
weighted avg	0.78	0.76	0.76	82

Figure 46: Metrics of SVM Gaussian Model on df with label 2 and 3 merged post SMOTE

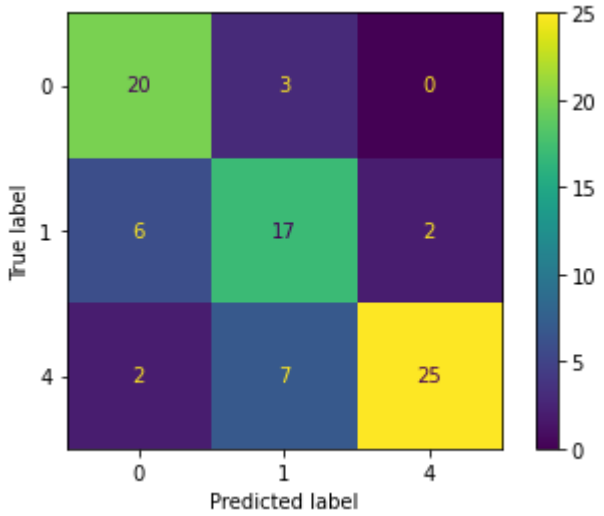


Figure 47: Confusion Matrix of SVM Gaussian Model on df with label 2 and 3 post SMOTE

	precision	recall	f1-score	support
0	0.70	0.91	0.79	23
1	0.52	0.52	0.52	25
4	0.85	0.68	0.75	34
accuracy			0.70	82
macro avg	0.69	0.70	0.69	82
weighted avg	0.71	0.70	0.69	82

Figure 48: Metrics of SVM Sigmoid Model on df with label 2 and 3 merged post SMOTE

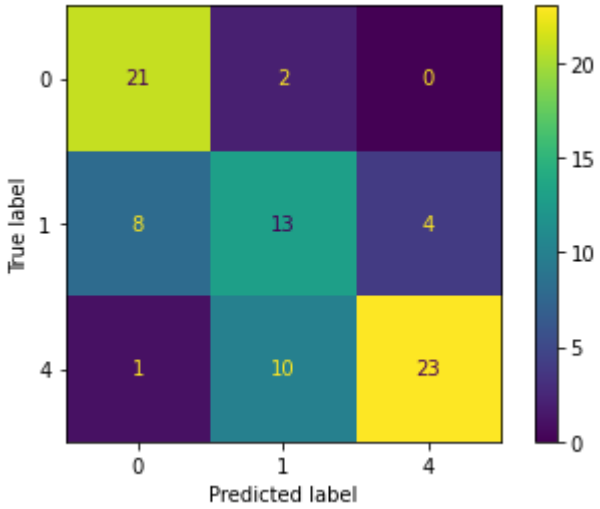


Figure 49: Confusion Matrix of SVM Sigmoid Model on df with label 2 and 3 post SMOTE

5.1.3 Cross Validation on df post SMOTE with label 2 and 3 merged

We applied parameters of 5-fold CV on precedent models.
These parameters are:

```
{'C': 100, 'gamma': 0.01, 'kernel': 'rbf'}
```

Figure 50: Parameters of 5-fold CV on df with label 2 and 3 merged post SMOTE

We implemented model according to Cross Validation:

	precision	recall	f1-score	support
0	0.65	0.87	0.74	23
1	0.54	0.60	0.57	25
4	0.96	0.65	0.77	34
accuracy			0.70	82
macro avg	0.71	0.71	0.69	82
weighted avg	0.74	0.70	0.70	82

Figure 51: Metrics of SVM Gaussian Model on df with label 2 and 3 merged post SMOTE and with 'C': 100, 'gamma': 0.01

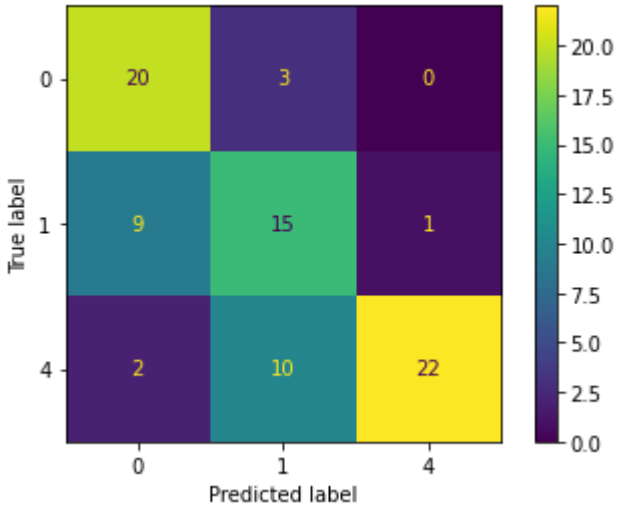


Figure 52: Confusion Matrix of SVM Gaussian Model on df with label 2 and 3 merged post SMOTE and with 'C': 100, 'gamma': 0.01

5.2 SVM without SMOTE

To give a qualitative dimension of the impact of SMOTE, we re-run algorithms seen previously without the oversampling. As we can see, with a gaussian and sigmoid kernel, machine learning algorithm were not able to detect the fourth label (Moderate Dementia) because of its low numerosity, so it made predictions with only three labels. Outcomes are anyway quite poor, and that suggests to us the relevance of data augmentation to train good models. Results for linear, polynomial, gaussian and sigmoid kernels are shown from here:

	precision	recall	f1-score	support
0	0.65	0.87	0.74	23
1	0.48	0.56	0.52	25
4	0.95	0.62	0.75	34
accuracy			0.67	82
macro avg	0.69	0.68	0.67	82
weighted avg	0.72	0.67	0.68	82

Figure 53: Metrics of SVM Linear Model on df without SMOTE

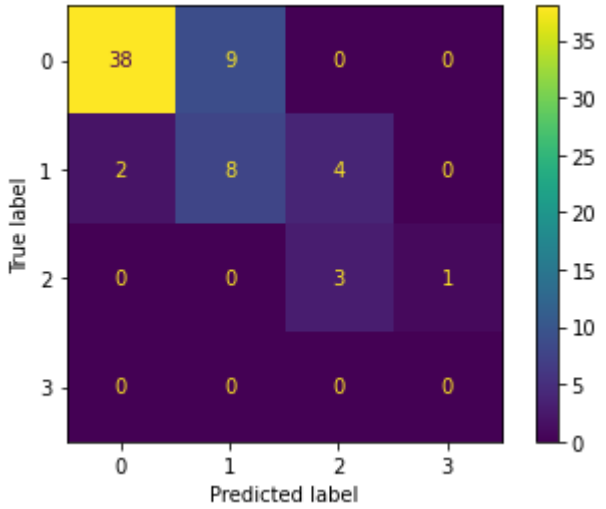


Figure 54: Confusion Matrix of SVM Linear Model on df without SMOTE

	precision	recall	f1-score	support
0	0.77	0.94	0.85	47
1	0.33	0.07	0.12	14
2	0.50	0.50	0.50	4
3	0.00	0.00	0.00	0
accuracy			0.72	65
macro avg	0.40	0.38	0.37	65
weighted avg	0.66	0.72	0.67	65

Figure 55: Metrics of SVM Polynomial Model on df without SMOTE

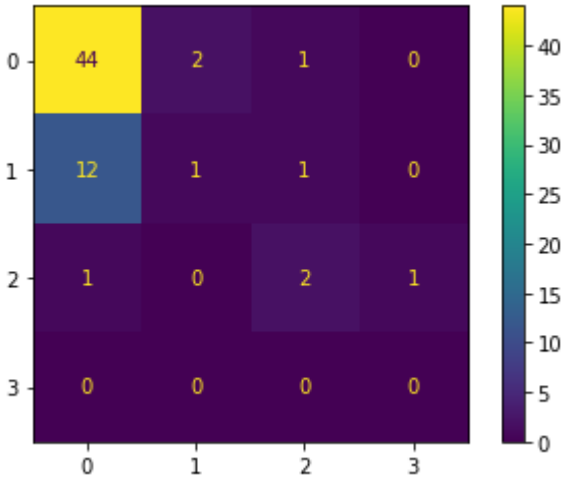


Figure 56: Confusion Matrix of SVM Polynomial Model on df without SMOTE

	precision	recall	f1-score	support
0	0.88	0.91	0.90	47
1	0.43	0.43	0.43	14
2	0.00	0.00	0.00	4
accuracy			0.75	65
macro avg	0.44	0.45	0.44	65
weighted avg	0.73	0.75	0.74	65

Figure 57: Metrics of SVM Gaussian Model on df without SMOTE

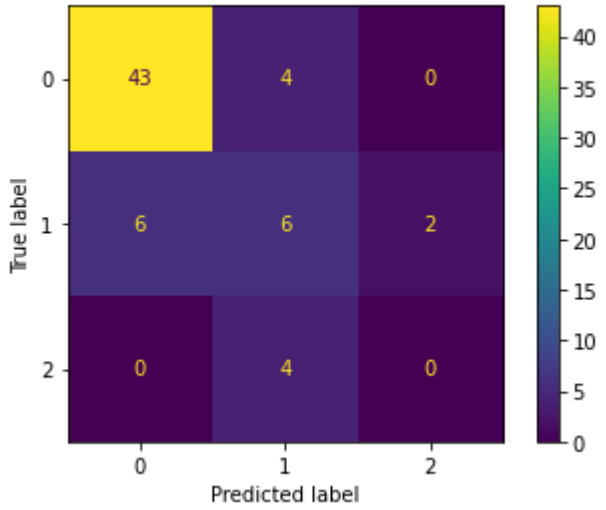


Figure 58: Confusion Matrix of SVM Gaussian Model on df without SMOTE

	precision	recall	f1-score	support
0	0.90	0.98	0.94	47
1	0.69	0.64	0.67	14
2	1.00	0.25	0.40	4
accuracy			0.86	65
macro avg	0.86	0.62	0.67	65
weighted avg	0.86	0.86	0.85	65

Figure 59: Metrics of SVM Sigmoid Model on df without SMOTE

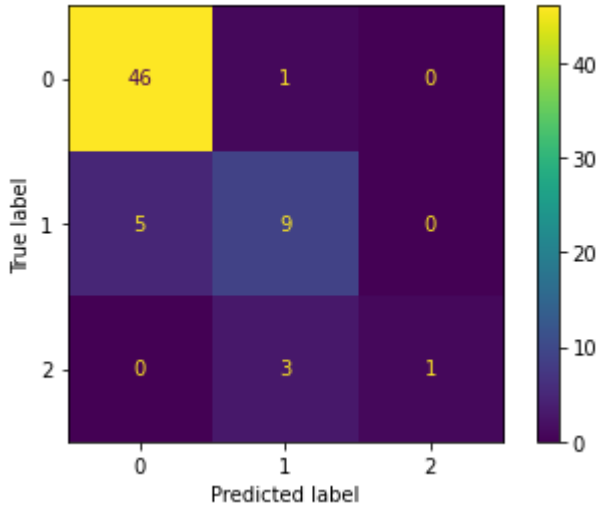


Figure 60: Confusion Matrix of SVM Sigmoid Model on df without SMOTE

5.2.1 Cross Validation on starting df

In respect of what has been done before, we made a 5-fold CV to optimize parameters. Best parameters are:

```
{'C': 1, 'gamma': 0.1, 'kernel': 'sigmoid'}
```

Figure 61: Parameters of 5-fold CV on df without SMOTE

So, we re built an SVM with sigmoid kernel, C equals to 1 and gamma equals to 0.1, and, as we can see below, results are very satisfactory.

	precision	recall	f1-score	support
0	0.88	0.94	0.91	47
1	0.67	0.57	0.62	14
2	1.00	0.75	0.86	4
accuracy			0.85	65
macro avg	0.85	0.75	0.79	65
weighted avg	0.84	0.85	0.84	65

Figure 62: Metrics of SVM Sigmoid Model on df without SMOTE with C = 1 and gamma = 0.1

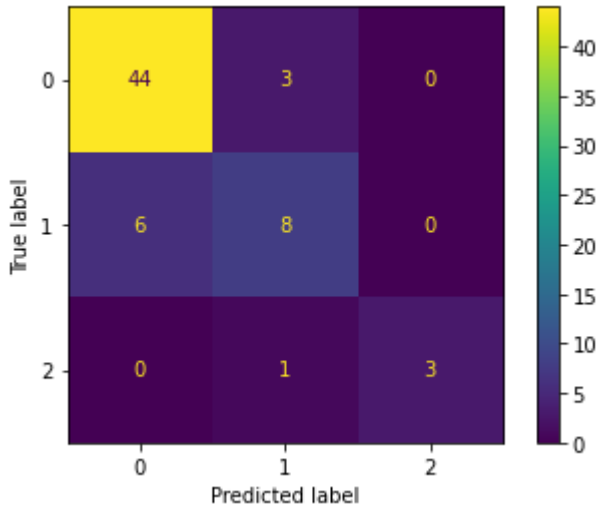


Figure 63: Confusion Matrix of SVM Sigmoid Model on df without SMOTE with C = 1 and gamma = 0.1

5.2.2 Merging Mild Dementia and Moderate Dementia on starting df

We built the same models with label 'Mild Dementia' and 'Moderate Dementia' merged.

	precision	recall	f1-score	support
0	0.78	0.97	0.87	37
1	0.36	0.31	0.33	16
4	0.80	0.33	0.47	12
accuracy			0.69	65
macro avg	0.65	0.54	0.56	65
weighted avg	0.68	0.69	0.66	65

Figure 64: Metrics of SVM Linear Model on df without SMOTE with label 2 and 3 merged

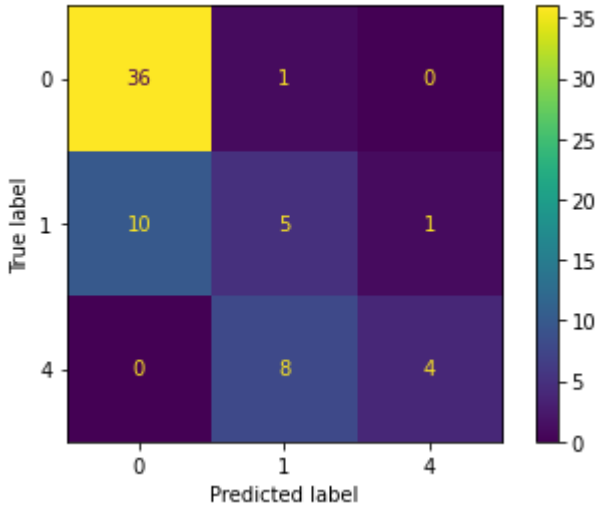


Figure 65: Confusion Matrix of SVM Linear Model on df with label 2 and 3 merged

	precision	recall	f1-score	support
0	0.60	1.00	0.75	37
1	0.00	0.00	0.00	16
4	0.50	0.08	0.14	12
accuracy			0.58	65
macro avg	0.37	0.36	0.30	65
weighted avg	0.43	0.58	0.45	65

Figure 66: Metrics of SVM Polynomial Model on df without SMOTE with label 2 and 3 merged

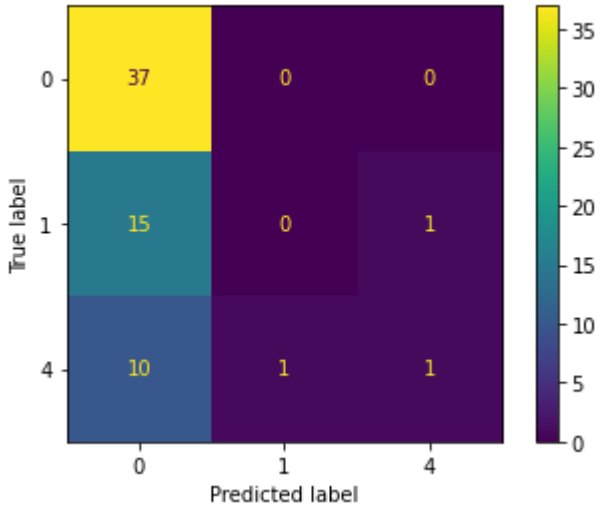


Figure 67: Confusion Matrix of SVM Polynomial Model on df with label 2 and 3 merged

	precision	recall	f1-score	support
0	0.70	0.95	0.80	37
1	0.20	0.19	0.19	16
4	0.00	0.00	0.00	12
accuracy			0.58	65
macro avg	0.30	0.38	0.33	65
weighted avg	0.45	0.58	0.51	65

Figure 68: Metrics of SVM Gaussian Model on df without SMOTE with label 2 and 3 merged

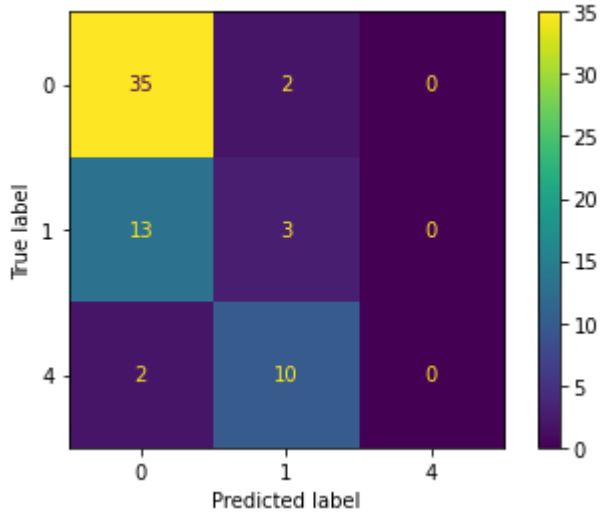


Figure 69: Confusion Matrix of SVM Gaussian Model on df with label 2 and 3 merged

	precision	recall	f1-score	support
0	0.74	0.95	0.83	37
1	0.22	0.25	0.24	16
4	0.00	0.00	0.00	12
accuracy			0.60	65
macro avg	0.32	0.40	0.36	65
weighted avg	0.48	0.60	0.53	65

Figure 70: Metrics of SVM Sigmoid Model on df without SMOTE with label 2 and 3 merged

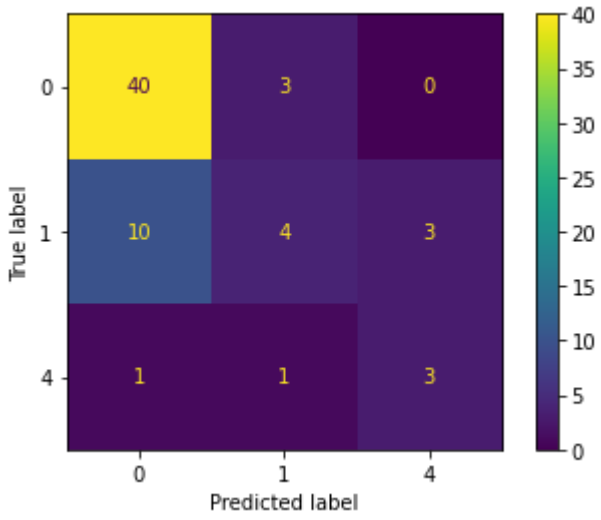


Figure 71: Confusion Matrix of SVM Sigmoid Model on df with label 2 and 3 merged

5.2.3 Cross Validation on starting df with label 2 and 3 merged

We made 5-fold Cross Validation here too. These are parameters:

```
{'C': 0.5, 'gamma': 0.1, 'kernel': 'sigmoid'}
```

Figure 72: Parameters of 5-fold CV on df without SMOTE and label 2 and 3 merged

So we re-run SVMs for sigmoid kernel with C equals to 0.5 and gamma equals to 0.1.

	precision	recall	f1-score	support
0	0.74	0.95	0.83	37
1	0.22	0.25	0.24	16
4	0.00	0.00	0.00	12
accuracy			0.60	65
macro avg	0.32	0.40	0.36	65
weighted avg	0.48	0.60	0.53	65

Figure 73: Metrics of SVM Sigmoid Model on df without SMOTE with label 2 and 3 merged and C = 0.5 and gamma = 0.1

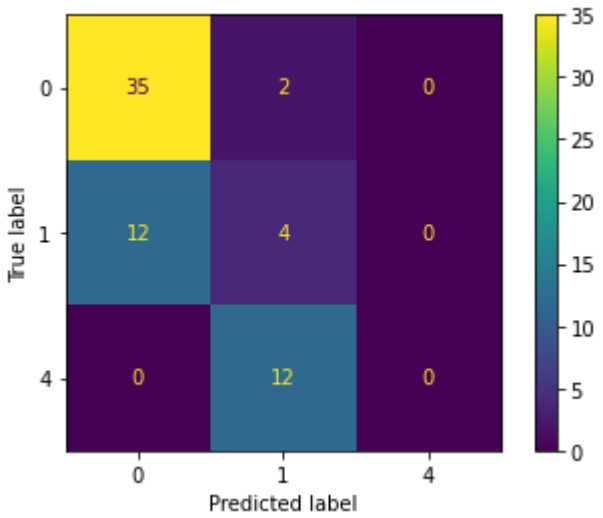
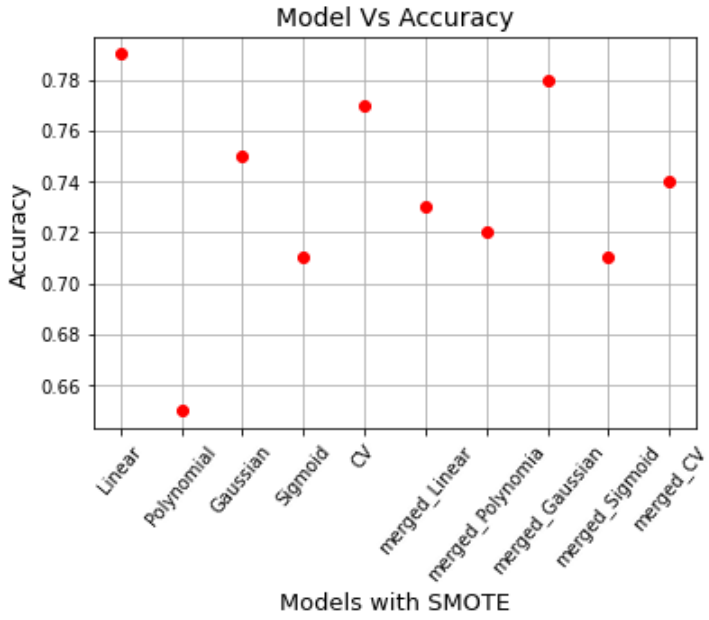


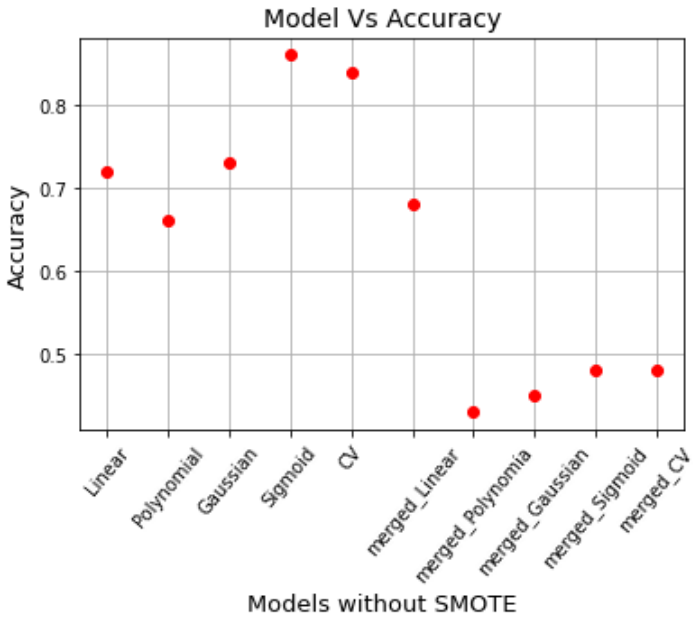
Figure 74: Confusion Matrix of SVM Sigmoid Model on df with label 2 and 3 merged and C = 0.5 and gamma = 0.1

Model is the best for its category but certainly can't be used for classificate or predict Alzheimer's so it cannot be considered.

6 Comparison among Machine Learning models

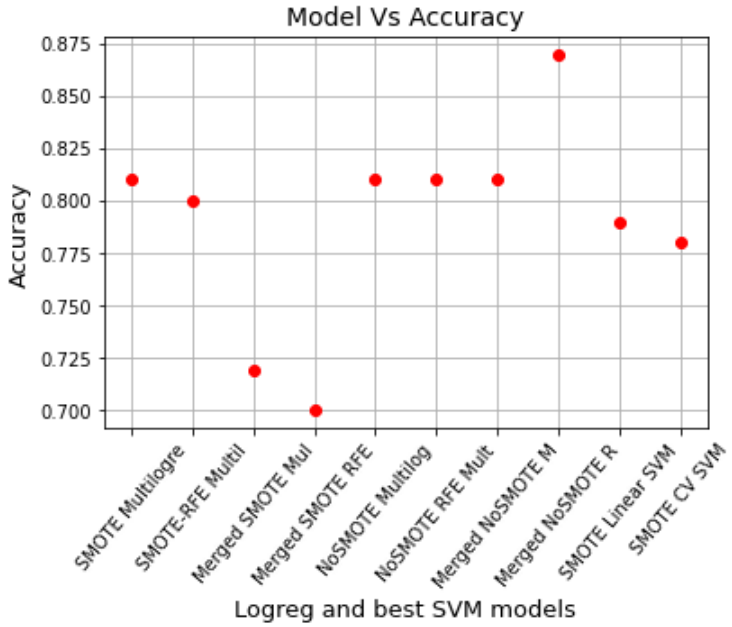
In next two charts we plotted accuracy per SVM model with SMOTE, in first graph, and without SMOTE in the second one.





SMOTE seems to improve general performance since average accuracy is higher than average accuracy without SMOTE. In particular, better model with SMOTE is

the one with linear kernel. Among SVMs without SMOTE stand out Sigmoid and CV without merged label, but if we see their Confusion Matrices we can notice that these model were not able to discriminate between Mild Dementia and Moderate Dementia, so we can't believe they are reliable. In fact, if we see accuracies of models with labels 2 and 3 merged, we realize they are quite small. At this point we compared best SVMs models with Logistic models. We deliberately avoided make a comparison with Ensamble models having previously rejected them.



Multi Logistic Regression on dataset post SMOTE with Recursive Feature Selection and label 2 and 3 merged soars as the model with higher accuracy, but this outcome is influenced by label 4 that is combination of Mild Dementia and Moderate Dementia. In this plot worst models are logistic ones with merged labels, while the other 'merged' models give good results. But if we want to choose a model that combines good performances and the possibily to classificate each label, we are prone to prefer Logistic Regression model with SMOTE and without RFE (because one with RFE gives an accuracy slightly lower). Other model, instead, are very valid if we are not interested in diversify between Mild Dementia and Moderate Dementia otherwise if we have a subclassifier able to distinguish between them. We will talk about it in section 7.

7 A subclassifier to discriminate Mild and Moderate AD

Many models that we built in previous section are very valid but can't differentiate between Mild and Moderate Dementia, so we started to think to a subclassifier applicable as a second step of one of the previous model. This challenge was very absorving because it meant build a model besed on few data, in particular observations of Moderate Dementia were just two. Because of this so small number, we decided to not applicate SMOTE or other data augmentation algorithm because they would have compromised our analysis and 'pushed' results too much towards that data. Always considering the scarcity of data and the unbalanced ratio (24/2), we chose to not use any Data Mining algorithm and to make an Exploratory Analysis based on boxplots of each feature to discover hidden patterns. For this analysis we dropped 'ID', 'Hand', 'Delay' features. We started this analysis observing summaries of two labels.

	M/F	Age	Educ	SES	MMSE	CDR	eTIV	nWBV	ASF
count	24.000000	24.000000	24.000000	24.000000	24.000000	24.0	24.000000	24.000000	24.000000
mean	0.291667	78.458333	2.583333	2.875000	21.958333	2.0	1477.791667	0.705500	1.194625
std	0.464306	6.560615	1.380506	1.295897	3.263489	0.0	118.267927	0.030625	0.093525
min	0.000000	69.000000	1.000000	1.000000	15.000000	2.0	1274.000000	0.655000	1.013000
25%	0.000000	72.750000	1.750000	2.000000	20.000000	2.0	1397.500000	0.683750	1.161500
50%	0.000000	78.000000	2.000000	3.000000	22.000000	2.0	1477.000000	0.698000	1.188500
75%	1.000000	83.000000	4.000000	4.000000	23.000000	2.0	1511.500000	0.731500	1.256000
max	1.000000	96.000000	5.000000	5.000000	28.000000	2.0	1732.000000	0.762000	1.377000

Figure 75: Summary of individuals with Mild Dementia

	M/F	Age	Educ	SES	MMSE	CDR	eTIV	nWBV	ASF
count	2.000000	2.000000	2.000000	2.000000	2.0	2.0	2.000000	2.00000	2.000000
mean	0.500000	82.000000	2.000000	3.500000	15.0	3.0	1456.500000	0.68400	1.207000
std	0.707107	5.656854	1.414214	0.707107	0.0	0.0	78.488853	0.02687	0.065054
min	0.000000	78.000000	1.000000	3.000000	15.0	3.0	1401.000000	0.66500	1.161000
25%	0.250000	80.000000	1.500000	3.250000	15.0	3.0	1428.750000	0.67450	1.184000
50%	0.500000	82.000000	2.000000	3.500000	15.0	3.0	1456.500000	0.68400	1.207000
75%	0.750000	84.000000	2.500000	3.750000	15.0	3.0	1484.250000	0.69350	1.230000
max	1.000000	86.000000	3.000000	4.000000	15.0	3.0	1512.000000	0.70300	1.253000

Figure 76: Summary of individuals with Moderate Dementia

For the difficulty to make comparisons looking these tables we plotted boxplots of features next to each other, as to observe differences. Let's start with Age variable (we exclude M/F because, for its nature, it does not have sense make a boxplot).

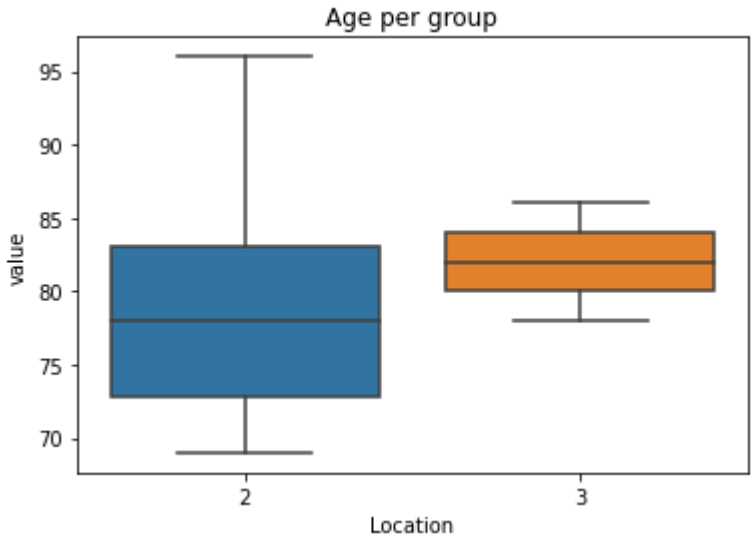


Figure 77: Comparison of Age boxplots

As we can see, median value of Moderate Dementia is higher than the Mild Dementia one, but we are prone to not consider this information because there are age values of the first class sharply higher the maximum of Moderate Dementia. Clearly, in this case is apparent that low numerosity of data does not allow us to make conclusions.

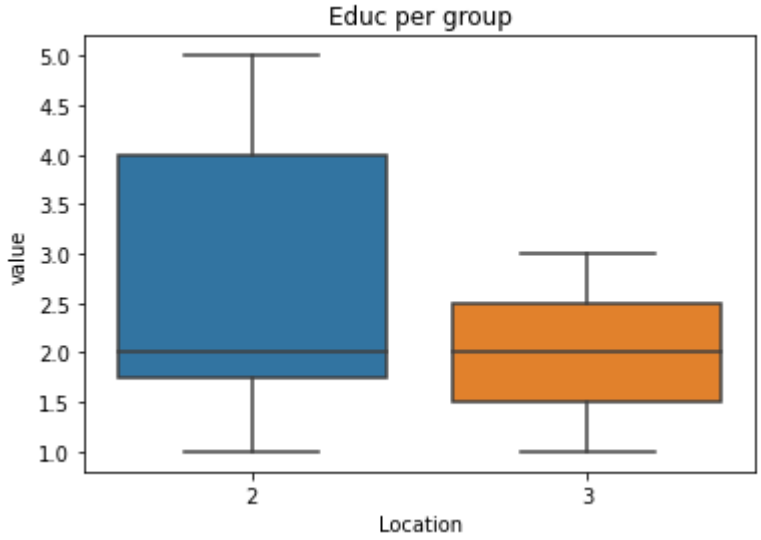


Figure 78: Comparison of Educ boxplots

Median level of education is the same for both labels, but Mild Dementia belong to individuals with every degree of education (concentration is however toward the bottom).

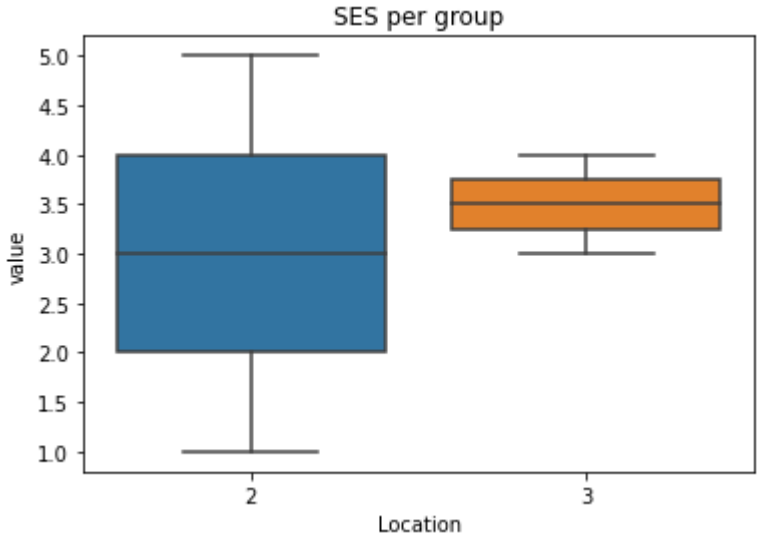


Figure 79: Comparison of SES boxplots

SES level is higher for individuals with Moderate Dementia. This data could be due to more opportunities for assistance and higher life expectancy, but without appropriate data we can not validate this statement, that may perhaps depend only from case.

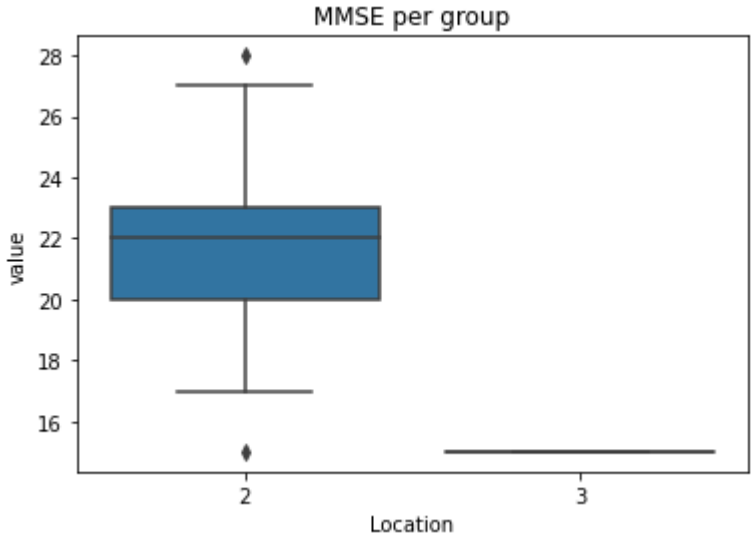


Figure 80: Comparison of MMSE boxplots

Mini-Mental State Examination score is indicative of the importance of surveys on patients to detect AD's, in fact the two individuals with Moderate Dementia are marked with lowest MMSE score.

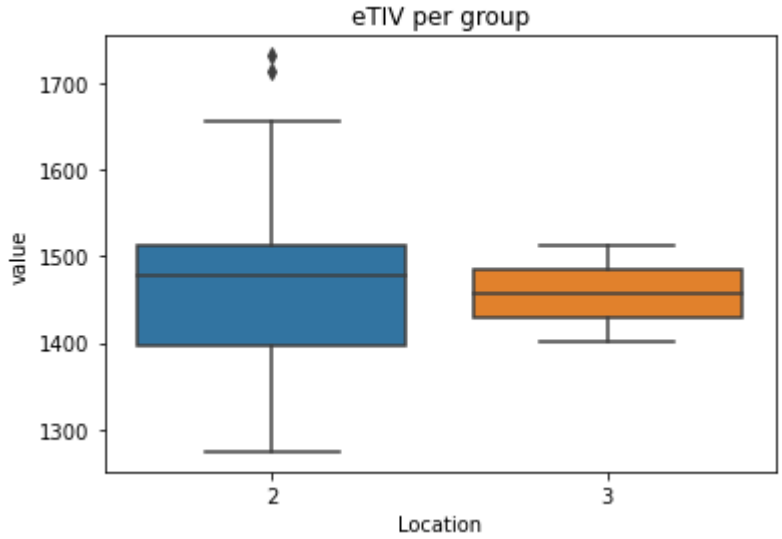


Figure 81: Comparison of eTIV boxplots

Dimension of estimated total intracranial volume is broadly centred for both labels, which can not be said for the normalized whole brain volume (shown below).

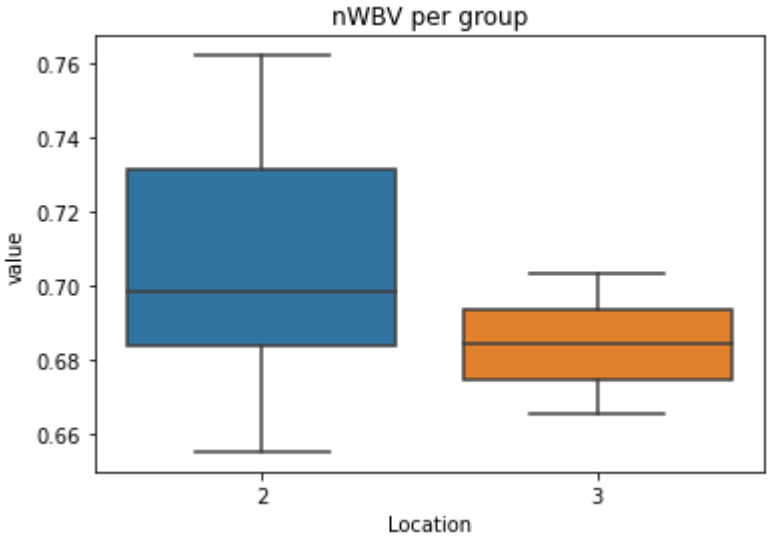


Figure 82: Comparison of nWBV boxplots

Finally, boxplots of ASF.

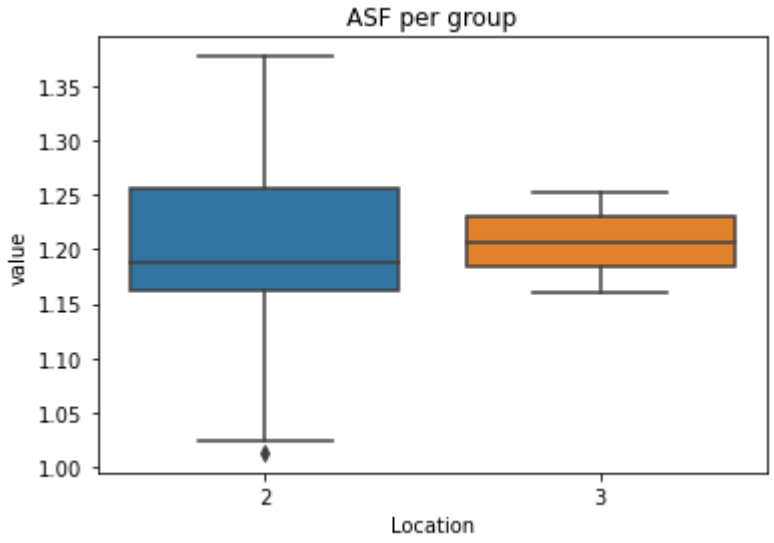


Figure 83: Comparison of ASF boxplots

In this case, range of Mild Dementia for this scaling factor is very large, in contrast to range of Moderate Dementia one which is very smaller.
In conclusion, this analysis based on boxplots didn't lead us to anything important and this was due to low numerosity of data that was crucial in our not being able to create a subclassifier. Unfortunately, in our opinion is necessary have more data to build a double classifier model (model seen in precedence + subclassifier), or, alternatively, use a model like the SMOTE Multilogistic Regression or SMOTE Multilogistic Regression post RFE, if it is desired a lighter model, to predict AD's with a good degree of accuracy.

8 Convolutional Neural Network for Image Processing

The second part of this notebook focuses on developing and analyzing a CNN, a Convolutional Neural Network[18]. The purpose is the same: classify the different phases of Alzheimer's Disease, but this time we use structural MRI images of brains. The model was built using an open source deep-learning tool called TensorFlow[19], this allows us to easily build, train, optimize and then evaluate the majority

[18]https://en.wikipedia.org/wiki/Convolutional_neural_network
[19]https://www.tensorflow.org/

of neural networks architectures. There are also more tools like PyTorch, but we decided to use TensorFlow for its easiness and immediacy.

For this part of the work we used a dataset found on Kaggle.
Kaggle is an interesting place for Data Science lovers to share and work on data. The dataset consists of 6400 bidimensional BW images with a standard shape of (208,176) saved into 4 different folders, each containing a sample of data for each category to identify:

- Mild Demented
- Moderate Demented
- Non Demented
- Very Mild Demented

To import this dataset we utilized a small function that loads and opens each file of each folder and saves the output in an empty list, associating the correct lable. We did the operation for each label and for test and train parts. Here is the output:

```
Lenghts of Train data are, in order:

2560 1792 717 52
Lenghts of Test data are, in order:

640 448 179 12
```

Figure 84: Output from the code

To have a general idea about the dataset content, and what we're working with, we plotted an image for each label. Visualizing data is important, we want to see what the Neural Network will see. Here is what our data looks like:

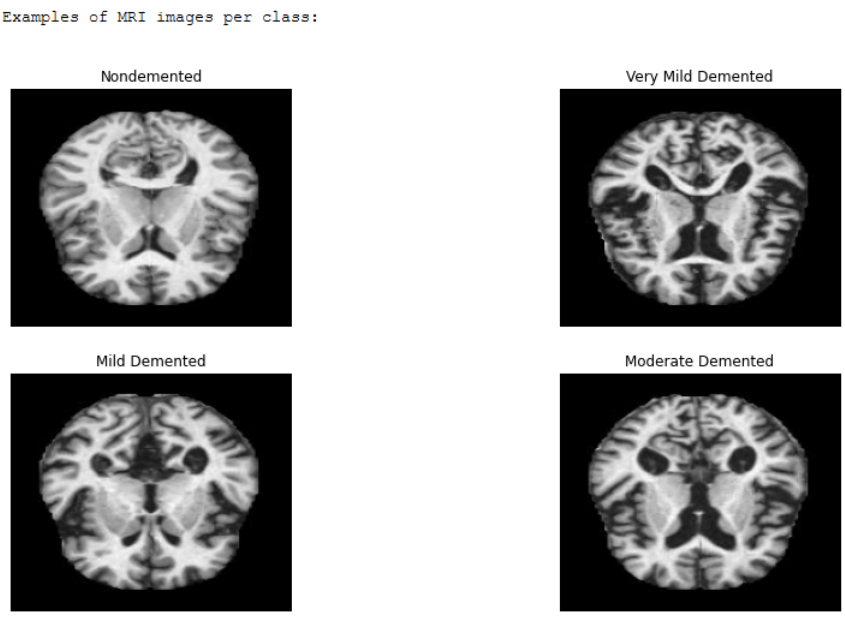


Figure 85: MRI scans of brains

The first thing we do is to merge the datasets with pandas to obtain the entire set. This is our whole dataset:

```
Shape of finale dataframe is:
(6400, 2)
```

Balancing, for supervised learning, is an important factor to consider. The underepresentation of one or more classes could be a potential problem for the model, leading to poor performances. In this case the daset is strongly unbalanced, with 3200 images in the Non-Demented class, 2240 images in the VeryMild-Demented class, 896 in the Mild-Demented class and just 64 images in the Moderate-Demented class. The first step to fix this issue is to normalize data, we used a MinMax function for this matter:

```
def normalization(array):
    '''This function normalizes train data'''
    train_norm = []
    transformer = MinMaxScaler()
    for value in array:
        value = transformer.fit_transform(value)
        train_norm.append(value)
    return train_norm

train_norm = normalization(train_data)
```

Figure 86: Code for the MinMax fucntion

Secondly, we are working with categorical data. A good way to make the work easier for the model is to convert our labels into arrays of 0s and 1s. This process is known as One-Hot encoding and helps the model understand the type of data it is working with.

```
from sklearn.preprocessing import LabelBinarizer

onehot = LabelBinarizer()
labels = onehot.fit_transform(labels)
print(labels)
```

```
[[0 0 1 0]
 [0 0 1 0]
 [0 0 1 0]
 ...
 [0 1 0 0]
 [0 1 0 0]
 [0 1 0 0]]
```

Figure 87: Code and output for One-Hot function.

Now it's crucial to perform the split into test and train dataset. While usually the proportion should be train being 20 percent of the whole dataset, it's important to mantain the split between all different length of all variables. After performing the split here's how our dataset looks like:

```
length X_train: 5120
length X_test: 1280
length y_train: 5120
length y_test: 1280
```

Figure 88: Our dataset split into test and train

The last step of our preprocessing phase is to try to remove unbalancing. Applying class-weights can easily solve the issue. The function returned weights for each class, so we could move on the next part of the work, building the model. Here are the weights for the classes:

```
Dictionary of weights:
{0: 1.7852161785216178, 1: 25.098039215686274, 2: 0.5, 3: 0.7142857142857143}
```

Figure 89: Class weights output

At this point, we built the model using Tensorflow.
We created a Sequential model because it is appropriate for a plain stack of layers where each layer has exactly one input tensor and one output tensor[20].
More specifically, The sequential structure of the model is composed by several different layers:

1. an initial input layer;
2. some hidden layers, to define our convolutional structure as a Deep Neural Network (DNN);
3. a final output layer.

The convolutional neural network is a specialized type of neural network model designed for working with two-dimensional image data, although they can be used with one-dimensional and three-dimensional data. Central to the convolutional neural network is the convolutional layer that gives the network its name. This layer performs an operation called a "convolution". In the context of a convolutional neural network, a convolution is a linear operation that involves the multiplication of a set of weights with the input, much like a traditional neural network. Given that the technique was designed for two-dimensional input, the multiplication is performed between an array of input data and a two-dimensional array of weights, called a filter or a kernel[21]. Another parameter that we setted arbitrarily was activation function, id est a function that defines the output of a node given an input or set of inputs. We used relu activation function, defined as the positive part of its argument:

$$f(x) = x^+ = \max(0, x) \tag{1}$$

We also added other kind of layers, in particular, two of them were MaxPooling2D ones. They[22] downsample the input representation by taking the maximum value over the window defined by pool size for each dimension along the features axis. The window is shifted by strides in each dimension.The resulting output shape when using the "same" padding option is:

$$outputshape = math.floor[(inputshape - 1)/strides] + 1 \tag{2}$$

We also added Dense layers to make our neural network densely-connected. Finally, we apply dropouts and flatten input. Summary of built model is show below:

[20] https://keras.io/guides/sequentialmodel/
[21] https://machinelearningmastery.com/convolutional-layers-for-deep-learning-neural-networks/
[22] https://keras.io/api/layers/pooling$_layers/max$pooling2d/

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 104, 88, 64)       1664
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 52, 44, 64)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 26, 22, 128)       204928
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 13, 11, 128)       409728
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 7, 6, 256)         819456
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 3, 3, 256)         0
_________________________________________________________________
flatten (Flatten)            (None, 2304)              0
_________________________________________________________________
dense (Dense)                (None, 64)                147520
_________________________________________________________________
dropout (Dropout)            (None, 64)                0
_________________________________________________________________
dense_1 (Dense)              (None, 32)                2080
_________________________________________________________________
dropout_1 (Dropout)          (None, 32)                0
_________________________________________________________________
dense_2 (Dense)              (None, 4)                 132
=================================================================
Total params: 1,585,508
Trainable params: 1,585,508
Non-trainable params: 0
```

Figure 90: Summary of CNN

Before launching algorithms to fit model, we chose parameters that would be the most appropriate to help the model to achieve the best performance.The first step was to decide the metric that will be used to evaluate the model. Considering the unbalancing, overall accuracy does not seem the best metric to use for this task, but we kept it for its easy interpretation. We added the area under the ROC curve (AUC) as a second metric. AUC provides an aggregate measure of performance across all possible classification thresholds. One way of interpreting AUC is as

the probability that the model ranks a random positive example more highly than a random negative example.
Among hyperparameters, the first hyperparameter to control is the learning rate. Selecting a wrong value of learning rate could strongly affect the model performances, leaving it stuck in a local minima or in a plateau if the value is too low, or just jumping on the other side of the global minima if the value is too high. This is why it is so important to try different value of learning rate. Here we used 1e-4. Due to the fact that we are working with categorical variables, the model is compiled using 'categorical crossentropy' as loss function. I then randomly selected the optimizer (RMSprop) and the activation functions of the layers (Relu).
We used callbacks in order to optimize the time of the training. In particular the early stopping. This method allows to optimize the model monitorizing the progress toward the converging point, stopping the training when the pick is reached even if the number of epochs selected is not completed.
Finally, to have more reliable results, we decided to use a 3-fold cross validation process. The cross validation process allows us to train the model k times, each time resampling the validation set. In this way we can see if the model performances are close to each other and it makes our results more robust. Let's see results:

```
Val_Acc Folder 1:  0.98046875
Val_Acc Folder 2:  0.966796875
Val_Acc Folder 3:  0.974609375
--------------------------------
Val_Auc Folder 1:  0.9967060089111328
Val_Auc Folder 2:  0.9944502711296082
Val_Auc Folder 3:  0.994667649269104
```

Figure 91: Results fot each fold

Our model seems to achieved the best result of 0.98 accuracy on the validation in each folder of cross validation. This means that, at least on paper (we still need to test it), the model works well overall. Furthermore, also the AUC score is high, and this make our results even stronger. Usually this is the moment where, looking at our results on the validation set, we can tune our hyperparameters to further improve the model performance. Despite the importance of this step,considering the results obtained, the hyperparametes optimization would be a waste of time.
After the training we can create a visual representation of the model's performance focusing our attention on the train/val accuracy trend and the train/val loss.

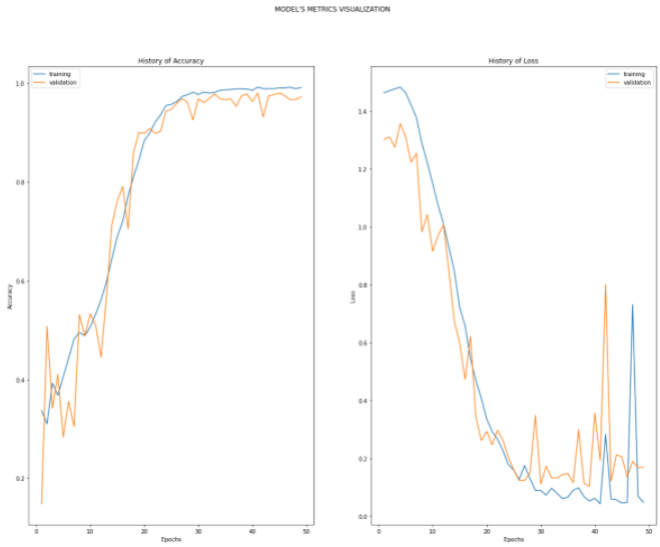


Figure 92: Train/Val accuracy and Train/Val loss for model 1

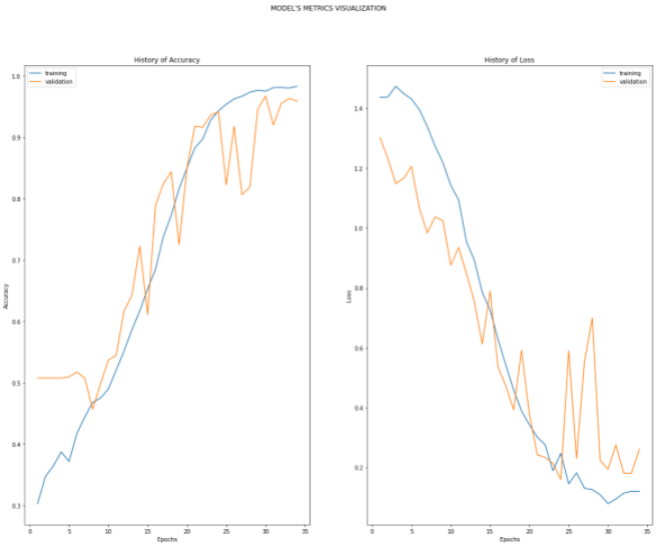


Figure 93: Train/Val accuracy and Train/Val loss for model 2

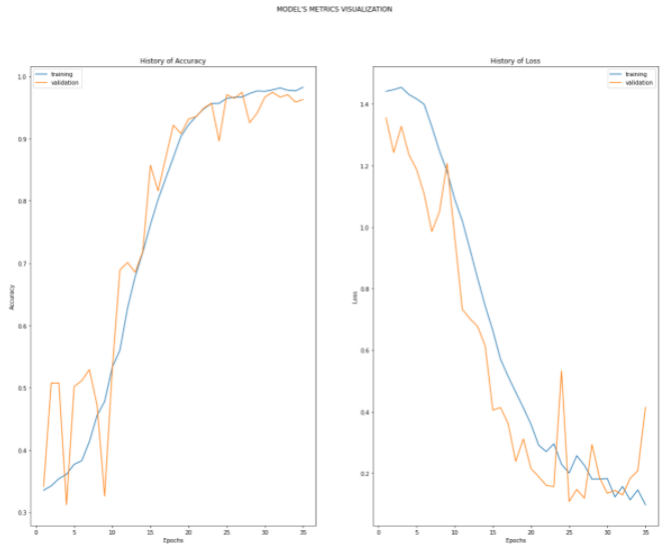


Figure 94: Train/Val accuracy and Train/Val loss for model 3

Looking results, best model is the first, so we used it to predict on the test set previously created.
The plotting of ROC and Precision-Recall curve is a further tool to see how our model is performing in the classification task. The first plot represents a graph showing the performance at all classification thresholds using two parameters: True Positive Rate and False Positive Rate. On the other side, the precision-recall curve shows the tradeoff between precision and recall for different threshold. A high area under the curve represents both high recall and high precision, where high precision relates to a low false positive rate, and high recall relates to a low false negative rate. A further advantage of the precision-recall curve is that, differently from the ROC curve, considers the class imbalance. The ROC[23] curve is created by plotting the true positive rate (TPR) against the false positive rate (FPR) at various threshold settings. The true-positive rate is also known as sensitivity, recall or probability of detection in machine learning.

[23] https://en.wikipedia.org/wiki/Receiver$_{o}perating_{c}haracteristic$

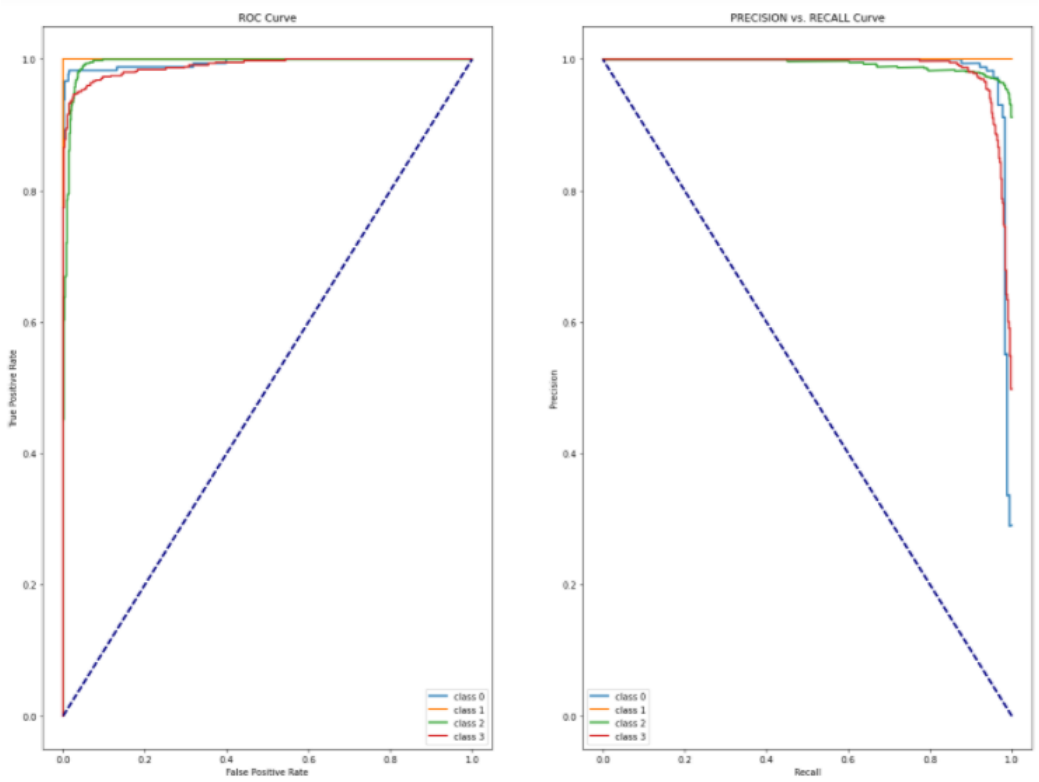


Figure 95: ROC and PRECISION vs. RECALL curves

where precision:

$$\mathrm{PPV} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}} = 1 - \mathrm{FDR} \tag{3}$$

and recall:

$$\mathrm{TPR} = \frac{\mathrm{TP}}{\mathrm{P}} = \frac{\mathrm{TP}}{\mathrm{TP} \dashv \mathrm{FN}} = 1 - \mathrm{FNR} \tag{4}$$

Then, we showed confusion matrix. In the field of machine learning and specifically the problem of statistical classification, a confusion matrix[24], also known as an error matrix, is a specific table layout that allows visualization of the performance of an algorithm, typically a supervised learning one (in unsupervised learning it is usually called a matching matrix).

[24]https://en.wikipedia.org/wiki/Confusion$_matrix$

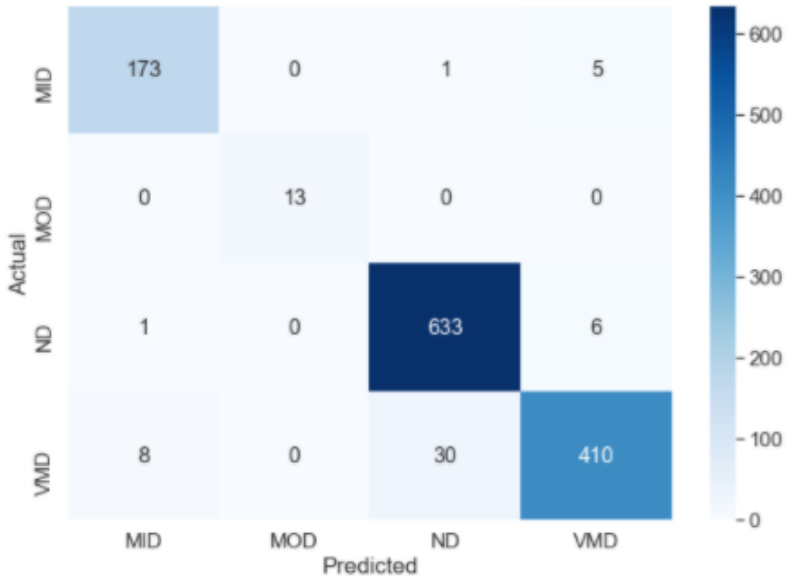


Figure 96: Confusion Matrix

where MD represents Mild Dementia, MOD Moderate Dementia, ND Non Dementia and VMD Very Mild Dementia.
To conclude the evaluation of our work we can plot the classification report, a table that shows the main metrics (Accuracy, Precision, Recall and F1Score) for each single class.

	precision	recall	f1-score	support
MID	0.95	0.97	0.96	179
MOD	1.00	1.00	1.00	13
ND	0.95	0.99	0.97	640
VMD	0.97	0.92	0.94	448
accuracy			0.96	1280
macro avg	0.97	0.97	0.97	1280
weighted avg	0.96	0.96	0.96	1280

```
{0: 0.9948802256962944, 1: 1.0, 2: 0.9936962890625, 3: 0.9898077996222527}
```

Figure 97: Classification Report

9 Pixels as features

Second method applied to images dataset was an experimental approach that we read on a paper proposed by professor D'Ambrosio[25]. Basically, it is about retrive pixel composition of every image in order to create a new dataset whose records are different images and pixel are variables. With our dataset this approach was very suitable because images had the same shape, so every pixel-feature has the same position in every image. In addition, composing block of pixel you get a lower computational complexity and a greater interpretative easiness.
In contrast to the approach proposed by professor D'Ambrosio, we worked without adding other known variables (like for example CDR) in training phase. So, necessarily dataset creation was first phase. We used to this scope glob, PIL, pandas and numpy libraries. In particular, from PIL we imported Image[26] that contains the command *.reduce* that returns a copy of the image reduced factor times. We choose to apply minimum reduction with a factor equals to 2.
To fight unbalancing, we used precedent weights, while, to make prediction we used classification trees. As criteria for calculating information gain we used Gini Index and Entropy both. Gini Index is calculated as:

$$\text{Gini} = 1 - \sum_{i=1}^{n} p^2\left(c_i\right) \tag{5}$$

and Entropy as:

$$\text{Entropy } = \sum_{i=1}^{n} -p\left(c_i\right) \log_2\left(p\left(c_i\right)\right) \tag{6}$$

Above is reported Confusion Matrix for one with Gini Index as splitting criterion:

$$\text{ConfusionMatrix} = \begin{bmatrix} 156 & 166 & 141 & 20 \\ 34 & 162 & 258 & 27 \\ 4 & 57 & 356 & 51 \\ 0 & 0 & 34 & 454 \end{bmatrix} \tag{7}$$

[25] Dynamic recursive tree-based partitioning for malignant melanoma identification in skin lesion dermoscopic images; Massimo Aria, Antonio D'Ambrosio, Carmela Iorio, Roberta Siciliano, Valentina Cozza

[26] https://pillow.readthedocs.io/en/stable/reference/Image.html

and this is Confusion Matrix with Entropy as splitting criterion:

$$\text{ConfusionMatrix} = \begin{bmatrix} 158 & 203 & 92 & 30 \\ 31 & 299 & 112 & 39 \\ 11 & 168 & 193 & 96 \\ 0 & 0 & 0 & 488 \end{bmatrix} \tag{8}$$

In both cases we reach an accuracy equals to 0.59. Let's see Classification Reports for both cases:

Report :	precision	recall	f1-score	support
0.0	0.80	0.32	0.46	483
1.0	0.42	0.34	0.37	481
2.0	0.45	0.76	0.57	468
3.0	0.82	0.93	0.87	488
accuracy			0.59	1920
macro avg	0.62	0.59	0.57	1920
weighted avg	0.63	0.59	0.57	1920

Figure 98: (Reduce 2) Classification Report for first Tree

Report :	precision	recall	f1-score	support
0.0	0.79	0.33	0.46	483
1.0	0.45	0.62	0.52	481
2.0	0.49	0.41	0.45	468
3.0	0.75	1.00	0.86	488
accuracy			0.59	1920
macro avg	0.62	0.59	0.57	1920
weighted avg	0.62	0.59	0.57	1920

Figure 99: (Reduce 2) Classification Report for second Tree

Results are not reliable so we have deepened the research trying to make a stronger reduction (with a factor equals to 40). These are new Confusion Matrices obtained:

$$\text{ConfusionMatrix} = \begin{bmatrix} 260 & 148 & 25 & 50 \\ 58 & 282 & 66 & 75 \\ 18 & 276 & 55 & 119 \\ 0 & 153 & 0 & 335 \end{bmatrix} \tag{9}$$

$$\text{ConfusionMatrix} = \begin{bmatrix} 287 & 0 & 167 & 29 \\ 103 & 0 & 316 & 62 \\ 37 & 0 & 319 & 112 \\ 10 & 0 & 153 & 335 \end{bmatrix} \tag{10}$$

In both cases we see poorest result (accuracies are $\sim$ 0,49) and under no circumstances we were able to predict Very Mild Dementia.
These are, instead, classification reports:

Report :	precision	recall	f1-score	support
0.0	0.77	0.54	0.63	483
1.0	0.33	0.59	0.42	481
2.0	0.38	0.12	0.18	468
3.0	0.58	0.69	0.63	488
accuracy			0.49	1920
macro avg	0.51	0.48	0.47	1920
weighted avg	0.52	0.49	0.47	1920

Figure 100: (Reduce 40) Classification Report for first Tree

Report :	precision	recall	f1-score	support
0.0	0.67	0.59	0.63	483
1.0	0.00	0.00	0.00	481
2.0	0.33	0.68	0.45	468
3.0	0.62	0.69	0.65	488
accuracy			0.49	1920
macro avg	0.41	0.49	0.43	1920
weighted avg	0.41	0.49	0.43	1920

Figure 101: (Reduce 40) Classification Report for second Tree

To conclude, though this experimental approach is very attractive can not be used to predict AD's.

10 Final Conclusions

As we can see, some models were very powerful and both with cross-sectional data and images we can classificate not only Alzheimer's Desease but also CDR with a degree of accuracy very high. As we saw with cross-sectional data one of th ebest model is Logistic Regression with SMOTE and RFE, and this result is very similar to one proposed by research team leaded by professor Jinhua Sheng in a paper[27] of the beginning of 2020. Best machine learning models were ones in which we merged Mild and Moderate Dementia, so to use them we we struggled to think how to create a sub-classifier with a small amount of data and as only solution we observed boxplots for each variable, without however obtaining results.
We have not found this issue in first Deep Learning model. The problematic of that model was the complexity of data preprocessing and difficulty of building the model. With Neural Network we have achieved excellent results and for this reason is the icing on the cake of our work (in which we discovered excellent models).
Finally last model proposed did not give us good results but for its seductiveness and simplicity we presented anyway.
In conclusion, this work is proposed only for Data Mining course. It is clear that to approve these models they need a check of a domain expert both for results and in construction phase.

[27] https://www.nature.com/articles/s41598-020-62378-0