Alzheimer's Disease Neuroimaging Initiative

Jin Cao, Jomel Meeko Manzano, Bowei Zhang

2024-02-22

Project Title:

Alzheimer's Disease Neuroimaging Initiative - Predicting Alzheimer's Disease in patients with Mild Cognitive Impairments

Investigator:

R.G. Thomas

Primary Analysts:

Jin Cao, j9cao@ucsd.edu
Jomel Meeko Manzano, jmmanzano@ucsd.edu
Bowei Zhang, boz013@ucsd.edu

Contents

Introduction

Mild cognitive impairment (MCI) is a condition in which people have more memory or thinking problems than other people their age. The symptoms of MCI are not as severe as those of Alzheimer's disease or a related dementia. People with MCI can usually take care of themselves and carry out their normal daily activities. People with MCI are at a greater risk of developing Alzheimer's disease or a related dementia.

Predicting Alzheimer's disease onset from Mild Cognitive Impairment (MCI) remains a challenge while many brain health-related data are becoming available (Grassi et.al, 2019). One example is the Alzheimer's Disease Neuroimaging Initiative(ADNI) dataset, It is a multicenter collaboration that collects data on MRI and PET images, genetics, cognitive tests, CSF, and blood biomarkers from Alzheimer's disease patients, mild cognitive impairment subjects, and elderly controls in a longitudinal format.

In this report, our goal is to develop a statistical model utilizing the ADNI dataset to predict whether a patient with characteristics of Mild Cognitive Impairment will progress to Alzheimer's Disease within 5 years. Our focus will be on patients at baseline, using this information to predict the likelihood of developing dementia within the next 5 years.

Data Stucture

Baseline Dataset

The original dataset 'ADNIMERGE' is a longitudinal dataset consisting of 2432 patients and 115 columns with 16443 rows. To obtain a dataset that fits our research objectives, we include participants that were diagnosed with EMCI or LMCI at the baseline. We also keep only the baseline measurement and add a new column 'converter' indicating if a patient is diagnosed with Dementia within 5 years since the baseline visit.

Completer Dataset

We also noticed that not every participants have entire 5-year of follow-up; therefore, we created another dataset called 'completer' that only includes participants with visit at month 60. The 'completer' dataset contains 270 observations, a noticeable reduction in number compared to the 'baseline' dataset.

We split both 'baseline' and 'completer' datasets into train and test sets by 70:30 ratio.

Main Analyses

Logistic Regression

The main model of choice in this study is the logistic regression for its robustness in dealing with all kinds of distribution of the predictors and the easiness of interpretation and inference. After fitting the full model with all available variables in the training dataset, a backward selection is conducted using the step function with AIC as the criterion. seven predictors are left in the model after the variable selection, and they all show statistical significance. The logistic regression model has achieved a 0.81 accuracy rate predicting the testing dataset.

Table 1: Logistic Regression on the baseline dataset

	Estimate	Std. Error	z value	Pr(>|z|)
(Intercept)	-2.6055	0.8824	-2.9527	0.0031
PTEDUCAT	0.0742	0.0356	2.0855	0.0370
CDRSB	0.3527	0.1201	2.9369	0.0033
FAQ	0.1124	0.0278	4.0461	0.0001
ADASQ4	0.1840	0.0502	3.6647	0.0002
RAVLT.immediate	-0.0302	0.0133	-2.2746	0.0229
LDELTOTAL	-0.1675	0.0342	-4.9020	0.0000
TRABSCOR	0.0037	0.0014	2.5899	0.0096

```
## [1] "Test Accuracy: 0.81"
```

Sensitivity Analysis

Logistic Regression: Completers Only

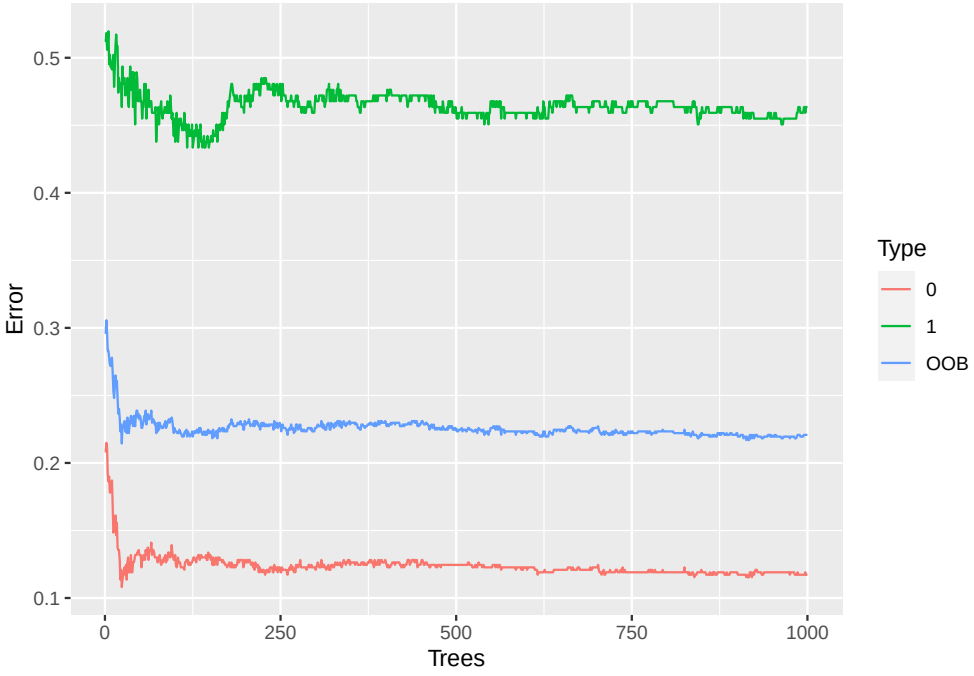
Table 2: Logistic Regression on the Completer Dataset

	Estimate	Std. Error	z value	Pr(>|z|)
(Intercept)	-5.8728	1.9299	-3.0431	0.0023
PTEDUCAT	0.1669	0.0795	2.1000	0.0357
CDRSB	1.0009	0.3151	3.1758	0.0015
FAQ	0.0756	0.0598	1.2645	0.2061
ADASQ4	0.4613	0.1119	4.1231	0.0000
RAVLT.immediate	0.0020	0.0254	0.0792	0.9369
LDELTOTAL	-0.2194	0.0796	-2.7566	0.0058
TRABSCOR	-0.0013	0.0042	-0.3135	0.7539

```
## [1] "Test Accuracy: 0.85"
```

Random Forest

In this section, we will discuss the Random Forest Model that we employed. The model was built on the training set using all available variables in the dataset. Initially, we experimented with 1000 trees, but eventually settled on 500 trees, which is the default setting in R. To evaluate the model's predictive performance, we examined the Out of Bag Error Rate. Notably, after employing 500 trees, the error rate did not show further significant decrease



Another tuning parameter we considered was the MTRY value, which governs the randomness in the variable selection process for each split in the construction of trees in a random forest. Initially, we fitted separate

models with MTRY values ranging from 1 to 10. Our analysis led us to the conclusion that a value of 4 resulted in the lowest Out of Bag (OOB) error.

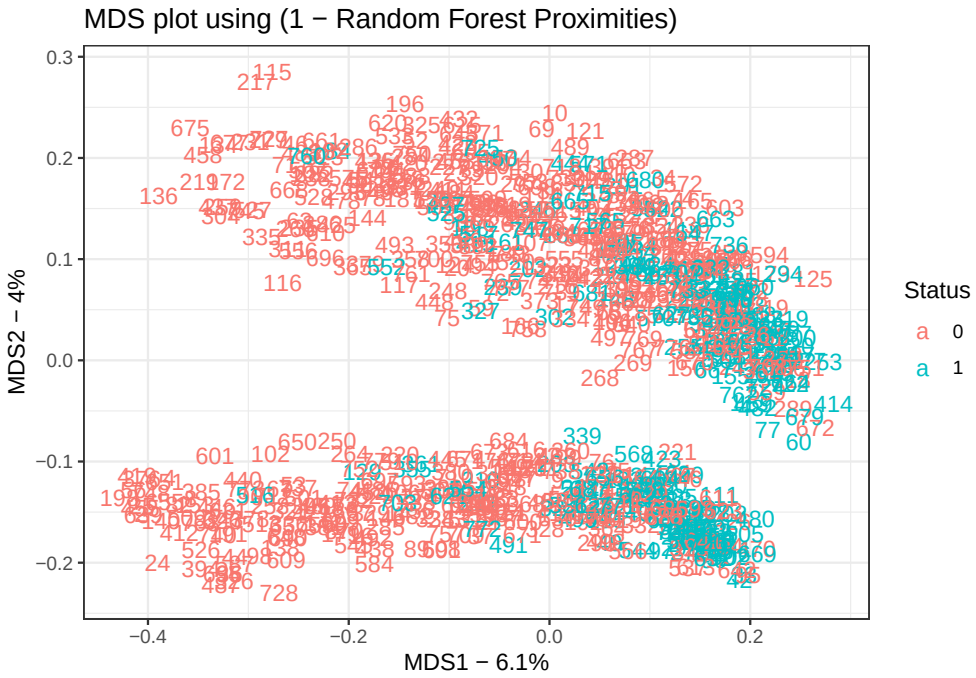
Table 3: OOB Error Rate for Different MTRY Values

MTRY	OOB Error
1	0.216
2	0.222
3	0.218
4	0.222
5	0.221
6	0.217
7	0.218
8	0.231
9	0.240
10	0.228

After configuring these tuning parameters, our final Random Forest model exhibited an Out of Bag (OOB) error rate of 22.34%. This implies that 77.02% of the OOB samples were correctly classified, as depicted in the confusion matrix below. Subsequently, we applied this model to predict the test set, achieving a prediction accuracy of 80%. Notably, the model seems to face challenges in correctly classifying converters, with an accuracy of 45%.

	0	1	class.error
0	502	44	0.0805861
1	124	109	0.5321888

For visual clarity, we generated a Multidimensional Scaling (MDS) plot to represent our model. In the plot, the non-converter group displays a broader spread, whereas the converter group forms a cluster. However, it's worth noting that some converters are scattered throughout the plot, extending beyond the main converter cluster.




```
## [1] "Test Accuracy: 0.8"
```

Neural Network

We also tried a simple neural network model of only one hidden layer with 8 neurons. After comparing different width and depth of the neural network architecture, we found out that a simple neural network with of only one hidden layer with 8 neurons gave the relatively best result with test accuracy of 77.61%.

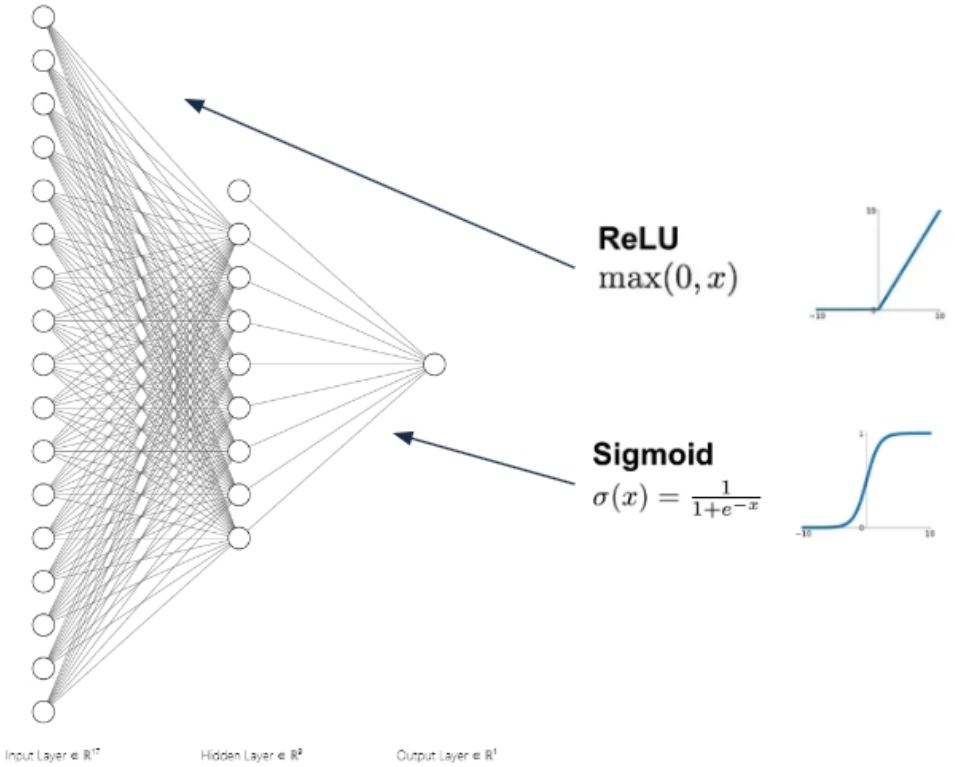


Figure 1: Neural Network: Architecture

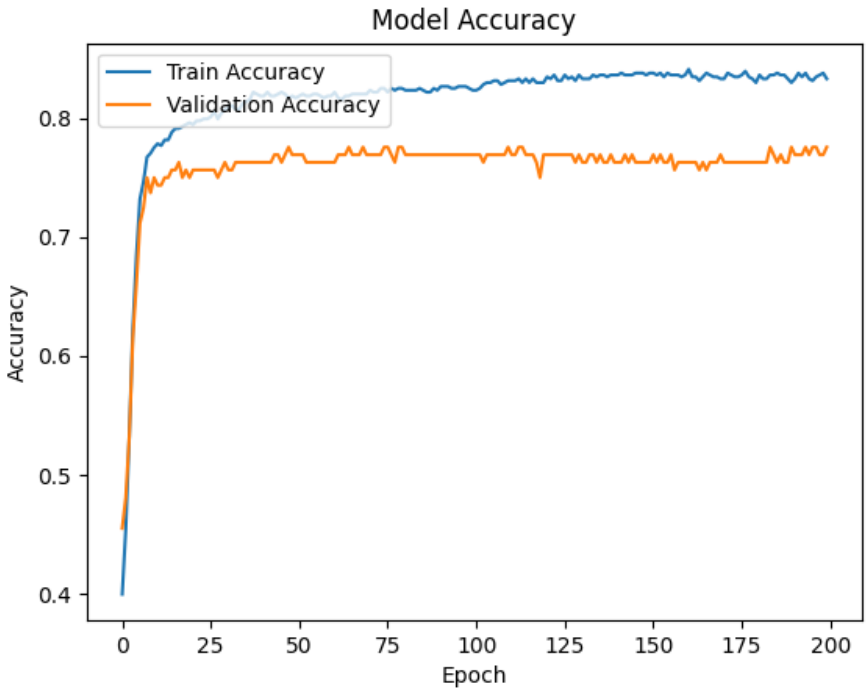


Figure 2: Traning and Validation Loss Curves

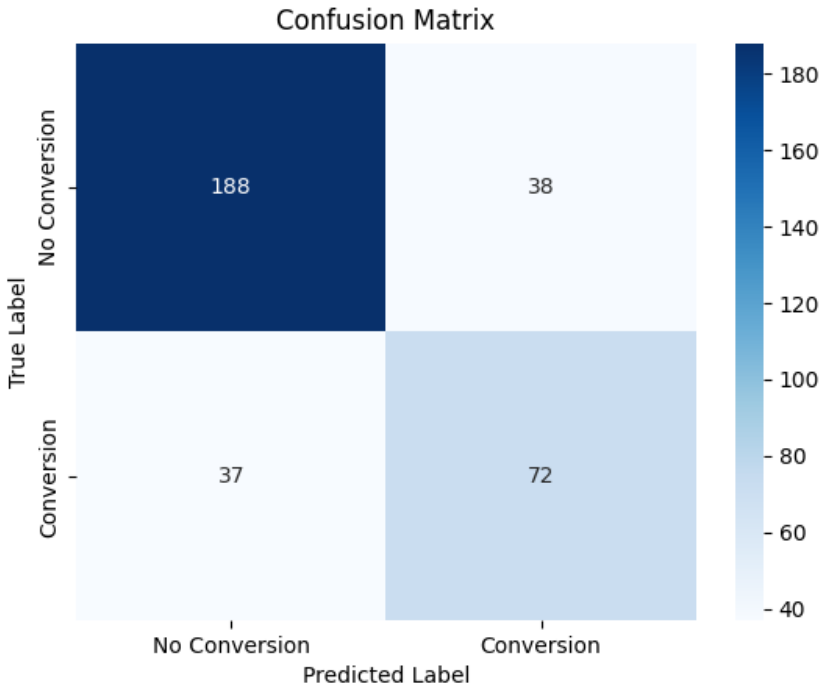


Figure 3: Confusion Matrix: Test set

Conclusion

Reference

Grassi, M., Rouleaux, N., Caldirola, D., Loewenstein, D., Schruers, K., Perna, G., ... & Alzheimer's Disease Neuroimaging Initiative. (2019). A novel ensemble-based machine learning algorithm to predict the conversion from mild cognitive impairment to Alzheimer's disease using socio-demographic characteristics, clinical information, and neuropsychological measures. Frontiers in neurology, 10, 756.

Van Buuren, S., & Groothuis-Oudshoorn, K. (2011). mice: Multivariate Imputation by Chained Equations in R. Journal of Statistical Software, 45(3), 1–67. https://doi.org/10.18637/jss.v045.i03

Code

```
knitr::opts_chunk$set(echo = TRUE)
Sys.setLanguage(lang = "en")
library(MASS)
library(readxl)
library(tidyverse)
library(caret)
test = read.csv("test.csv")
train = read.csv("train.csv")
test_completer = read.csv("test_completer.csv")
train_completer = read.csv("train_completer.csv")

fit1 = glm(converter ~ ., data = train, family = binomial)
fit2 = step(fit1, direction="backward", trace=FALSE)

s.fit2 = summary(fit2)
s.fit2$coefficients %>% round(4) %>% knitr::kable(caption = "Logistic Regression on the baseline dataset
predictions = predict(fit2, newdata = test, type = "response")
predictions_binary = ifelse(predictions > 0.5, 1, 0)
accuracy = predictions_binary == test$converter
accuracy_rate = sum(accuracy)/nrow(test)
print(paste0("Test Accuracy: ", round(accuracy_rate,2)))
fit3 = glm(formula = fit2$formula,
           data = train_completer,
           family = binomial)
s.fit3 = summary(fit3)
s.fit3$coefficients %>% round(4) %>% knitr::kable(caption = "Logistic Regression on the Completer Datas

predictions_s = predict(fit3, newdata = test_completer, type = "response")
predictions_s_binary = ifelse(predictions_s > 0.5, 1, 0)
accuracy_s = predictions_s_binary == test_completer$converter
accuracy_rate_s = sum(accuracy_s)/nrow(test_completer)

print(paste0("Test Accuracy: ", round(accuracy_rate_s,2)))
library(readxl)
library(ISLR2)
library(knitr)
library(class)
library(glmnet)
library(methods)
library(tree)
library(randomForest)
library(ggplot2)
library(gbm)
library(dplyr)
library(cowplot)

test = read.csv("test.csv")
train = read.csv("train.csv")
test_completer = read.csv("test_completer.csv")
train_completer = read.csv("train_completer.csv")
```

```r
# --- Variable Changes ---
# Specify the columns to be converted to factor
# Specify the columns to be converted to factor
char.var <- c("PTGENDER", "PTMARRY", "PTEDUCAT", "DX.bl")


train$PTGENDER <- as.factor(train$PTGENDER)
train$PTMARRY <- as.factor(train$PTMARRY)
#train$PTEDUCAT <- as.factor(train$PTEDUCAT)
train$DX.bl <- as.factor(train$DX.bl)
train$converter <- as.factor(train$converter)

# Test Set

test$PTGENDER <- as.factor(test$PTGENDER)
test$PTMARRY <- as.factor(test$PTMARRY)
#test$PTEDUCAT <- as.factor(test$PTEDUCAT)
test$DX.bl <- as.factor(test$DX.bl)
test$converter <- as.factor(test$converter)


# -------------- COMPLETERS -------------

train_completer$PTGENDER <- as.factor(train_completer$PTGENDER)
train_completer$PTMARRY <- as.factor(train_completer$PTMARRY)
#train_completer$PTEDUCAT <- as.factor(train_completer$PTEDUCAT)
train_completer$DX.bl <- as.factor(train_completer$DX.bl)
train_completer$converter <- as.factor(train_completer$converter)

# test_completer Set

test_completer$PTGENDER <- as.factor(test_completer$PTGENDER)
test_completer$PTMARRY <- as.factor(test_completer$PTMARRY)
#test_completer$PTEDUCAT <- as.factor(test_completer$PTEDUCAT)
test_completer$DX.bl <- as.factor(test_completer$DX.bl)
test_completer$converter <- as.factor(test_completer$converter)

set.seed(2024)

mtry.value <- floor(sqrt(ncol(train) - 1)) # Determine mtry

forest.adni <-  randomForest(converter ~ .,
                             mtry = mtry.value,
                             importance = TRUE,
                             proximity = TRUE,
                             ntree = 1000,
                             data = train)
# Optimality of Trees
oob.error.data <- data.frame(
  Trees=rep(1:nrow(forest.adni$err.rate), times=3),
  Type=rep(c("OOB", "0", "1"), each=nrow(forest.adni$err.rate)),
  Error=c(forest.adni$err.rate[,"OOB"],
    forest.adni$err.rate[,"0"],
```

```
    forest.adni$err.rate[,"1"]))

ggplot(data=oob.error.data, aes(x=Trees, y=Error)) +
  geom_line(aes(color=Type))
# MTRY Value search
oob.values <- vector(length=10)
for(i in 1:10) {
  temp.model <- randomForest(converter ~ .,
                             mtry = i,
                             importance = TRUE,
                             proximity = TRUE,
                             ntree = 500,
                             data = train)
  oob.values[i] <- temp.model$err.rate[nrow(temp.model$err.rate),1]
}

## find the optimal value for mtry...
which(oob.values == min(oob.values))

# Create a data frame
result_df <- data.frame(MTRY = 1:10,
                        OOB_Error = round(oob.values,3))
# Display the table using kable
kable(result_df, col.names = c("MTRY", "OOB Error"),
      caption = "OOB Error Rate for Different MTRY Values",
      format = "markdown")
forest.adni.final <-  randomForest(converter ~ .,
                             mtry = 1,
                             importance = TRUE,
                             proximity = TRUE,
                             ntree = 500,
                             data = train)

kable(forest.adni.final$confusion)

## Start by converting the proximity matrix into a distance matrix.
distance.matrix <- as.dist(1-forest.adni.final$proximity)

mds.stuff <- cmdscale(distance.matrix, eig=TRUE, x.ret=TRUE)

## calculate the percentage of variation that each MDS axis accounts for...
mds.var.per <- round(mds.stuff$eig/sum(mds.stuff$eig)*100, 1)

## now make a fancy looking plot that shows the MDS axes and the variation:
mds.values <- mds.stuff$points
mds.data <- data.frame(Sample=rownames(mds.values),
  X=mds.values[,1],
  Y=mds.values[,2],
  Status=train$converter)

ggplot(data=mds.data, aes(x=X, y=Y, label=Sample)) +
  geom_text(aes(color=Status)) +
  theme_bw() +
```

```
  xlab(paste("MDS1 - ", mds.var.per[1], "%", sep="")) +
  ylab(paste("MDS2 - ", mds.var.per[2], "%", sep="")) +
  ggtitle("MDS plot using (1 - Random Forest Proximities)")

predictions <- predict(forest.adni.final, newdata = test, type = "response")
predictions <- as.numeric(as.character(predictions))  # Convert to numeric if needed
predictions_binary <- ifelse(predictions > 0.5, 1, 0)
accuracy <- predictions_binary == test$converter
accuracy_rate <- sum(accuracy) / nrow(test)
print(paste0("Test Accuracy: ", round(accuracy_rate,2)))
```