# STA314: Alzheimer's Disease Diagnosis Report

**Abtin Shirvani**, 1006515610

Team Name: AShirvani

Rank: 16

Prediction: 0.92628

December, 2024

## 1 Problem Statement

### 1.1 Introduction

Alzheimer's Disease (AD) is a neurodegenerative disease primarily plaguing the elderly and it has been a growing public health concern, with prevalence expected to increase 3-fold by 2050.[1] The main reason why Alzheimer's Disease is such a big issue is because there are currently no cures, only licensed treatments to alleviate symptoms, and it gets exponentially worse with time.[2] Therefore, it is crucial that Alzheimer's Disease is diagnosed as early as possible to slow down cognitive decline.

### 1.2 Objective

The aim of this analysis is to develop a model that can accurately predict Alzheimer's Disease diagnosis using relevant information regarding the patient.

## 2 Statistical Analysis

### 2.1 Data pre-processing

The synthetic data was sourced from Kaggle and owned by Rabie El Kharoua.[3] The raw dataset contained 35 columns; 34 of which were features covering demographics, lifestyle factors, medical history, biomarkers, cognitive and functional assessments, and symptoms of the patients, while the last was the label of interest, AD Diagnosis. Patient ID and Doctor ID features were omitted as they intuitively contained no information regarding the diagnosis. Perhaps there is some argument that the doctor in charge played a role in the diagnosis, however their IDs were listed as confidential, so they would have been omitted regardless. Other than the doctor IDs, there were no missing values. The remaining features and diagnosis were formatted to their appropriate data types. Gradient Boosted Decision Trees (GBDTs), especially CatBoost, don't require much data preparation, so there were no further transformations to any of the features.

### 2.2 Model

Given predictive performance was the main goal, Gradient Boosting Decision Trees (GBDTs) were the natural choice. These types of algorithms are very well-suited for heterogeneous, tabular data and tend to perform better than other common ML methods (e.g. Logistic Regression, Random Forest, Support Vector Machines).[4] As mentioned earlier, GBDTs (or more generally Decision Trees) are very flexible in that they hold very little assumptions about the data; any monotonic transformation has no effect since we only care about the split point, thus they can handle raw data without the need

for scaling or normalization.[5] 5-fold Cross Validation using Log loss was favoured over accuracy as the metric for model performance when tuning the hyper-parameters due to the fact that accuracy doesn't account for the imbalanced diagnosis.[6]

## 2.3 Theory

The main idea of boosting is to start with a weak model, then fit another weak model that attempts to correct the errors of the previous model by using the previous model as an input. Repeating this process for enough iterations can lead to a surprisingly strong model. GBDTs are a specific type of boosting where the models being fit are decision trees and the errors from the previous models are calculated using the gradient of a loss function. The following in-depth summary of the process of training a GBDT for classification was sourced from Friedman's 1999 paper titled *Greedy function approximation: A gradient boosting machine.*[7]

### 2.3.1 Initialization

Initialize model with the optimal log-odds satisfying the following:

$$F_0(\mathbf{x}) = \underset{\gamma}{\operatorname{argmin}} \sum_{i=1}^{n} L(\gamma; y_i)$$

where $\gamma$ is the log-odds and $L(\gamma; y_i)$ denotes the loss function. In this case, the log-loss function was used, which is derived from the binomial likelihood:

$$\ell(p; y_1, ..., y_n) = \prod_{i=1}^{n} [p^{y_i} + (1-p)^{1-y_i}]$$

$$log(\ell(p; y_1, ..., y_n)) = \sum_{i=1}^{n} [y_i log(p) + (1-y_i) log(1-p)]$$

$$log(\ell(p; y_i)) = y_i log(\frac{p}{1-p}) - log(1-p) \qquad \forall\ i = 1, ..., n$$

We will express the log-likelihood in terms of the log-odds ($\gamma$) and take the negative to convert it to a minimization problem.

$$-log(\ell(p; y_i)) = L(\gamma; y_i) = y_i\gamma - log(1 + e^{\gamma}) \qquad \forall\ i = 1, ..., n$$

Plugging in the derived log-loss function, the initialization looks like the following:

$$F_0(\mathbf{x}) = \underset{\gamma}{\operatorname{argmin}} \sum_{i=1}^{n} [y_i\gamma - log(1 + e^{\gamma})]$$

### 2.3.2 Iterations

Now, we start computing the pseudo-residuals and fitting them to our regression trees iteratively, which we use to adjust each subsequent model. The pseudo-residuals are defined as:

$$r_{im} = -[\frac{\partial L(F(\mathbf{x}_i); y_i)}{\partial F(\mathbf{x}_i)}]_{F(\mathbf{x}) = F_{m-1}(\mathbf{x})}, \qquad \forall\ i = 1, ..., n$$

for $m$ number of iterations, where $\frac{\partial L(F(\mathbf{x}_i); y_i)}{\partial F(\mathbf{x}_i)}$ is the gradient of the loss function, computed using the previous iteration's log-odds estimates. $i$ refers to the sample number from the training data. The intuition behind taking the negative gradient of the loss function being that we move towards the minimum of the loss function. Next, we denote the leaves for the newly fitted regression tree as $R_{jm}$

for $j = 1, ..., J_m$ where $J_m$ is the total number of leaves, and compute a new log-odds ($\gamma_{jm}$) for each leaf.

$$\gamma_{jm} = \underset{\gamma}{\operatorname{argmin}} \sum_{i:\mathbf{x}_i \epsilon R_{jm}} L(F_{m-1}(\mathbf{x}_i) + \gamma; y_i)$$

This minimization problem is typically solved using 2nd order Taylor polynomial approximation to maintain convexity, whilst simplifying the computation. The steps are lengthy, however it leads to an elegant solution:

$$\gamma_{jm} = \frac{\sum_{i:\mathbf{x}_i \epsilon R_{jm}} r_{im}}{\sum_{i:\mathbf{x}_i \epsilon R_{jm}} p_i(1-p_i)}$$

where $p_i = \frac{e^{F_{m-1}(\mathbf{x}_i)}}{1+e^{F_{m-1}(\mathbf{x}_i)}}$, which denotes the probability of being class 1 for the i$^{\text{th}}$ sample derived from the previous iteration. The term in the numerator is the sum of the pseudo-residuals in leaf $j$. Finally, we can update our current iteration using the previous iteration's pseudo-residuals.

$$F_m(\mathbf{x}_i) = F_{m-1}(\mathbf{x}_i) + \nu \sum_{j=1}^{J_m} \gamma_{jm} I(\mathbf{x}_i \epsilon R_{jm})$$

where $\nu$ is the learning rate. The second term essentially sums up the pseudo-residuals that match the given $\mathbf{x}_i$, and its effect on the current iteration is shrunk by the learning rate. Apart from the initialization, this whole process repeats for $M$ iterations. The final classification is determined using the last predicted probabilities ($p_i = \frac{e^{F_M(\mathbf{x}_i)}}{1+e^{F_M(\mathbf{x}_i)}}$) given $\mathbf{x}_i$, with some cutoff threshold (typically 0.5).

$$\hat{y}_i = \begin{cases} 1 & p \geq 0.5 \\ 0 & otherwise \end{cases}$$

## 2.4 CatBoost

There are several extensions of GBDTs, the most popular being XGBoost for its efficiency and versatility. However, for the final model for this dataset, I opted for a less common GBDT algorithm called CatBoost (short for Categorical Boosting) since it tends to slightly outperform competing GBDT algorithms in terms of accuracy.[8]

### 2.4.1 Major Differences

The following section is sourced from a speech by Anna Veronika Dorogush (one of the creators of CatBoost) at the EuroPython 2018 conference, as well as the 2017 paper by Dorogush et al. titled *CatBoost: unbiased boosting with categorical features.*[9;10] There are 2 main characteristics differentiating CatBoost from other GBDT algorithms:

- how it builds regression trees for the pseudo-residuals
- how it handles categorical features

CatBoost builds symmetric regression trees. There are 2 advantages to this approach. Symmetric trees have been found to add more robustness with respect to the hyper-parameters, which reduces the need for excess time spent tuning the hyper-parameters. Secondly, they reduce the total runtime of the algorithm.

Normally for categorical features, one-hot encoding is used so the algorithm such as XGBoost can handle it, however there are some limitations, specifically for decision trees. One-hot encoding doesn't capture ordinality, so features such as Education would have to be converted to integers to be handled properly. This implies equal scaling between categories, which is unlikely to be the case. Also, when dealing with high cardinality features, it limits the amount of information captured in a split. For

instance, ethnicity could be split with *Caucasian* and *Black* on one side and *Asian* and *Other* on the other side, but with one-hot encoding it would be limited to a binary yes-no for one ethnicity. An alternative for encoding categorical features is Target Encoding, which essentially replaces each category with averages of the target variable for that category. The main issue with this method is that it makes the model prone to over-fitting because it is essentially using the information from the outcome to encode the feature (referred to as *target leakage*). CatBoost attempts to bypass this issue using ordered target encoding. The general idea is that it will assume an artificial order of the data (generated by a random permutation of the data) and take the average classification of each category in each categorical feature like in plain target encoding, but only for the samples before it, yielding the Target Statistic (TS). The TS ($\hat{x}_k^i$) of the $i^{\text{th}}$ feature and $k^{\text{th}}$ training example can be expressed by the following formulation:

$$\hat{x}_k^i = \frac{\sum_{\mathbf{x}_j=\mathcal{D}_k} I(x_j^i = x_k^i) \cdot y_i + ap}{\sum_{\mathbf{x}_j=\mathcal{D}_k} I(x_j^i = x_k^i) + a}$$

where $\mathcal{D}_k = \{\mathbf{x}_j : \sigma(j) < \sigma(k)\},\ \sigma$ is the permutation, $a > 0,\ p$ is the prior, and $x_j^i$ is the category of the $i^{\text{th}}$ feature and $j^{\text{th}}$ training example. In simpler terms, the numerator is taking the sum of all the previous training examples that:

1. share the same category as the current training example, and
2. have an observed classification of 1

The last term ($ap$) is to help smooth out the TS. The denominator denotes the sum of all the previous training examples that share the same category plus $a$. All together, the TS is essentially taking the average classification of a given category in the previous training examples. Different permutations are used for different iterations of gradient boosting to help keep variance low. For more details on the procedure, refer to Dorogush et al., 2017.[10]

#### 2.4.2 Hyper-parameters

CatBoost has 6 hyper-parameters that can be tuned:[11]

- **Depth**: number of splits for each decision tree
- **Learning rate**: adjusts how fast the model learns
- **Iterations**: max number of trees
- **L2 Regularization Coefficient**: adjusts the shrinkage of features
- **RSM**: proportion of features to use at each split in the trees
- **Border count**: number of splits for numerical features

## 3 Results

### 3.1 Exploratory Data Analysis

The final training dataset consisted of 32 features, of which roughly half were categorical, the other continuous. The categorical features other than gender were found to be heavily imbalanced. Participants in the dataset were mostly Caucasian and tended to exhibit no underlying health issues (including family history of Alzheimer's, cardiovascular disease, diabetes, hypertension, head injury, depression) and no symptoms (confusion, disorientation, personality changes, difficulty completing tasks, forgetfulness). Numerical features showed low levels of correlation (computed using Pearson and spearman coefficients).

| | Diagnosis | | |
|---|---|---|---|
| **Feature** | **Negative**, N = 972[1] | **Positive**, N = 532[1] | **p-value**[2] |
| **ADL** | 5.99 (3.40, 7.90) | 3.19 (1.48, 4.85) | <0.001 |
| **Behavioral Problems** | | | <0.001 |
| 0 | 887 (91%) | 389 (73%) | |
| 1 | 85 (8.7%) | 143 (27%) | |
| **Diabetes** | | | 0.041 |
| 0 | 803 (83%) | 461 (87%) | |
| 1 | 169 (17%) | 71 (13%) | |
| **Functional Assessment** | 6.30 (3.96, 8.23) | 3.35 (1.53, 4.93) | <0.001 |
| **Memory Complaints** | | | <0.001 |
| 0 | 860 (88%) | 335 (63%) | |
| 1 | 112 (12%) | 197 (37%) | |
| **Sleep Quality** | 7.20 (5.64, 8.58) | 6.87 (5.23, 8.44) | 0.014 |

[1] Median (IQR); n (%)

[2] Wilcoxon rank sum test; Pearson's Chi-squared test

Table 1: Distribution of positive and negative Alzheimer's diagnosis by features that showed a significant difference between groups ($p < 0.05$).

### 3.2 Model selection

A grid search was used to tune the hyper parameters, using the suggested ranges of values from the CatBoost documentation. The feature importance from the model output can be seen in Figure 1. We can see some correlation between the previous table summary and the feature importance graph. Functional assessment, ADL, and Memory complaints were ranked rather highly, suggesting that changing those feature values will heavily impact classification. However, it is important to keep the interpretation within the context of this specific model. It doesn't necessarily mean these features are in general more significant than the rest in determining diagnosis. Several alternative models were trained, dropping different "non-important" features measured from the graph. There were marginal differences in predictive performance, if any, and dropping more than 5 features began showing dips in performance. Also, the use of weights was explored to help deal with imbalanced classes of diagnosis. The recommended class weights $CW_k$ for the $k^{\text{th}}$ class is $\frac{\max_{c=1}^{K}(\sum_{t_i=c} w_i)}{\sum_{t_i=k} w_i}$, which given the initial weights are all 1, can be written more simply as $\frac{\max_{c=1}^{K}[\sum_{i=1}^{n} I(x_i=c)]}{\sum_{i=1}^{n} I(x_i=k)}$.[12] This essentially sums up the higher proportion class and divides by the number of samples in the $k^{\text{th}}$ class. Class weights for negative and positive diagnosis in this dataset are computed as 1 and $\approx 2$ respectively. Surprisingly, applying these class weights showed no differences in predictive performance.

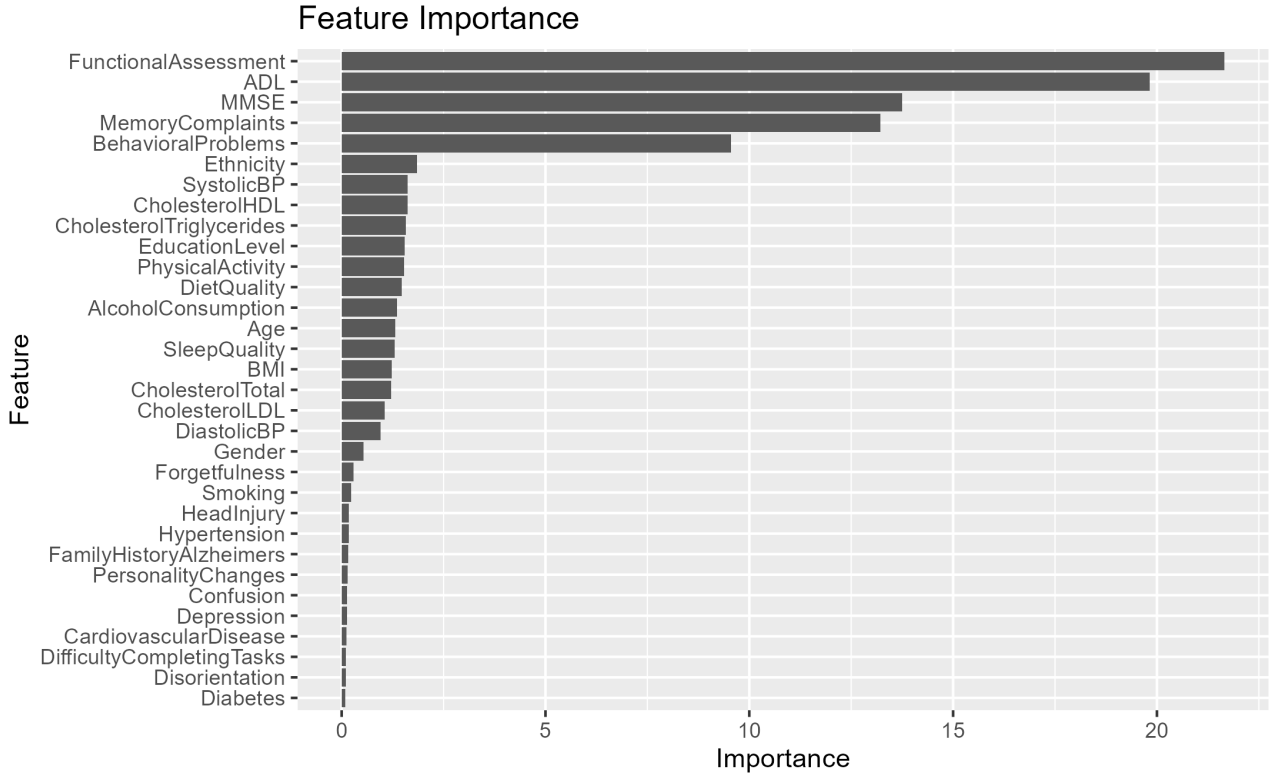


Figure 1: Feature Importance of the Final Model.

The final model included all the original features and unweighted classes with the grid search yielding the following hyper-parameter values:

- Depth: 10
- Learning rate: 0.01
- Iterations: 2000
- L2 Regularization Coefficient: 12
- RSM: 0.5
- Border count: 128

The final prediction score on Kaggle was 0.92628, meaning that 92.628% of the patients in that portion of the test dataset were diagnosed correctly.

# 4 Discussion

CatBoost in R lacked a lot of functionality compared to Python. One feature that I wanted to include was a visual for the decision trees and hyper parameter tuning, however I couldn't find a manageable way to do this for CatBoost in R. So for the future, I would lean more towards Python for ML analysis. Also in hindsight, I should have kept the class weights for the final model as I relied too heavily on the cross validation and initial Kaggle prediction score to gauge its efficacy. Comparisons between different algorithms and loss functions could also be considered. Moreover, the final model performance was decent, however there were others that far outperformed mine, suggesting that there is room for improvement for feature and model selection. I would like to explore more methods regarding this topic.

## References


[1] Deborah E Barnes and Kristine Yaffe. The projected effect of risk factor reduction on alzheimer's disease prevalence. *The Lancet Neurology*, 10(9):819–828, 2011. ISSN 1474-4422. doi: https://doi.org/10.1016/S1474-4422(11)70072-2. URL https://www.sciencedirect.com/science/article/pii/S1474442211700722.

[2] Clive Ballard, Serge Gauthier, Anne Corbett, Carol Brayne, Dag Aarsland, and Emma Jones. Alzheimer's disease. *The Lancet*, 377(9770):1019–1031, 2011. ISSN 0140-6736. doi: https://doi.org/10.1016/S0140-6736(10)61349-9. URL https://www.sciencedirect.com/science/article/pii/S0140673610613499.

[3] Rabie El Kharoua. Alzheimer's disease dataset, 2024. URL https://www.kaggle.com/dsv/8668279.

[4] Assaf Shmuel, Oren Glickman, and Teddy Lazebnik. A comprehensive benchmark of machine and deep learning across diverse tabular datasets, 2024. URL https://arxiv.org/abs/2408.14817.

[5] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An Introduction to Statistical Learning: with Applications in R*. Springer Publishing Company, Incorporated, 2014. ISBN 1461471370.

[6] Frank Harrell. Damage caused by classification accuracy and other discontinuous improper accuracy scoring rules. URL https://hbiostat.org/blog/post/class-damage/index.html.

[7] Jerome H. Friedman. Greedy function approximation: A gradient boosting machine. *The Annals of Statistics*, 29(5):1189 – 1232, 2001. doi: 10.1214/aos/1013203451. URL https://doi.org/10.1214/aos/1013203451.

[8] Candice Bentéjac, Anna Csörgo, and Gonzalo Martínez-Muñoz. A comparative analysis of gradient boosting algorithms. *Artificial Intelligence Review*, 54:1937 – 1967, 2019. URL https://api.semanticscholar.org/CorpusID:221283893.

[9] Anna Veronika Dorogush. Anna veronika dorogush - catboost - the new generation of gradient boosting, 2018. URL https://www.youtube.com/watch?v=oGRIGdsz7bM&ab_channel=EuroPythonConference.

[10] Liudmila Prokhorenkova, Gleb Gusev, Aleksandr Vorobev, Anna Veronika Dorogush, and Andrey Gulin. Catboost: unbiased boosting with categorical features. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, page 6639–6649, Red Hook, NY, USA, 2018. Curran Associates Inc.

[11] CatBoost. Parameter tuning, 2017. URL https://catboost.ai/docs/en/concepts/parameter-tuning.

[12] CatBoost. Training parameters, 2017. URL https://catboost.ai/docs/en/references/training-parameters/common#auto_class_weights.