Alzheimer's Disease Classification Project

8 December 2024

Pasut Aranchaiya	1011835935	Team name: US 3 with Igor Zakhidov
Zhanyan Guo	1008783130	Group number: Project 61
Igor Zakhidov	1007455549	Prediction score: 0.92628
Xiaomu Zhang	1009201089	Ranking: 18

1 Introduction

Alzheimer's disease (AD), is a neurodegenerative disorder that typically presents as memory loss, cognitive impairment, and the decline of activities of daily living(ADL). It has grown more serious in public health as the world's population ages, currently affecting 33.3% of people aged 85 or older [5]. As AD's risk increases exponentially after 65, early diagnosis will be crucial for treatment plans to slow its progression. However, the early symptoms of AD are often hidden, and the traditional diagnosis method generally relies on the doctor's experience and clinical tests, which are subjective, uncertain and often have the risk of delayed diagnosis and misdiagnosis. Improvement in early-stage diagnostic efficiency and accuracy will help create personalized treatment plans and may delay the progression of the disease, significantly improving the patient's quality of life. Therefore, it is essential to apply machine learning(ML) methods to refine and standardize the early diagnosis process. The further application of digital transformation in the medical field may also provide new insights for global public health management and disease prevention. By analyzing large-scale data, machine-learning models can discover new disease patterns, and potential risk factors, thereby advancing the development of disease research.

This study aims to identify the most important factors in AD's classification from a comprehensive dataset and to develop a predictive supervised ML modelling pipeline to classify individuals into two groups: those without AD's (benign = 0) and those with AD's (malignant = 1). This will be achieved by analyzing the patients' clinical values and comparing the performance of several commonly used models, such as Logistic Regression(LR) and Decision Trees(DT). The study mainly focuses on features of the 32 patients' clinical manifestations in the target database, such as age, gender, BMI, etc. In the future, this study might be expanded to facilitate early detection and even provide patients with personalized treatment plans.

2 Statistical Analysis

2.1 Dataset

The Alzheimer's Disease Kaggle dataset is provided to assist ML algorithms in detecting the early-stage AD. Additionally, this provides an opportunity to explore key features that may influence the occurrence of the disease. The dataset includes 1,504 patients with unique IDs for model training and 645 separate patients' data reserved for unbiased testing. Each patient is represented by 34 features, including medical history, lifestyle, demographics, and cognitive assessment fields, where only a subset of these features are considered relevant to AD and will be used in constructing our models.

2.2 Exploratory Data Analysis (EDA)

The dataset is determined to have no null entries and duplicates, meaning data cleaning is not necessary. We split the imbalanced dataset into training(80%) and validation(20%) sets, maintaining the original representative label class distribution in both, to prevent any data leakage of the validation set in EDA and model training. Then, irrelevant columns like "PatientID" and "DoctorInCharge" are dropped. To prevent any features from dominating the models' training process, the standard scaler was fit to the training features and both sets were standardized for magnitude-sensitive algorithms like LR. To visualize the linear relationships between every predictor feature pair and the label "Diagnosis", we plotted a heatmap (Figure 1) of the Pearson correlation coefficient matrix. It is obvious that 5 features, including the Mini-Mental State Examination "MMSE", "FunctionalAssessment", "MemoryComplaints", "BehavioralProblems", and clinical assessment of "ADL", are distinctively correlated with the label. This is expected as MMSE, Functional

Assessment, and ADL are measurable and reliable tests used by clinicians to classify AD, while behavioral problems and memory complaints are common symptoms and reactions of AD [5,6]. Geerlings et al. (1999) have even stated that the presence of memory complaints can be used to determine AD in its early stages as elderly people may notice changes in cognitive function before mental status tests are able to detect any problems. The above 5 correlations are the only relations that we deemed as important.

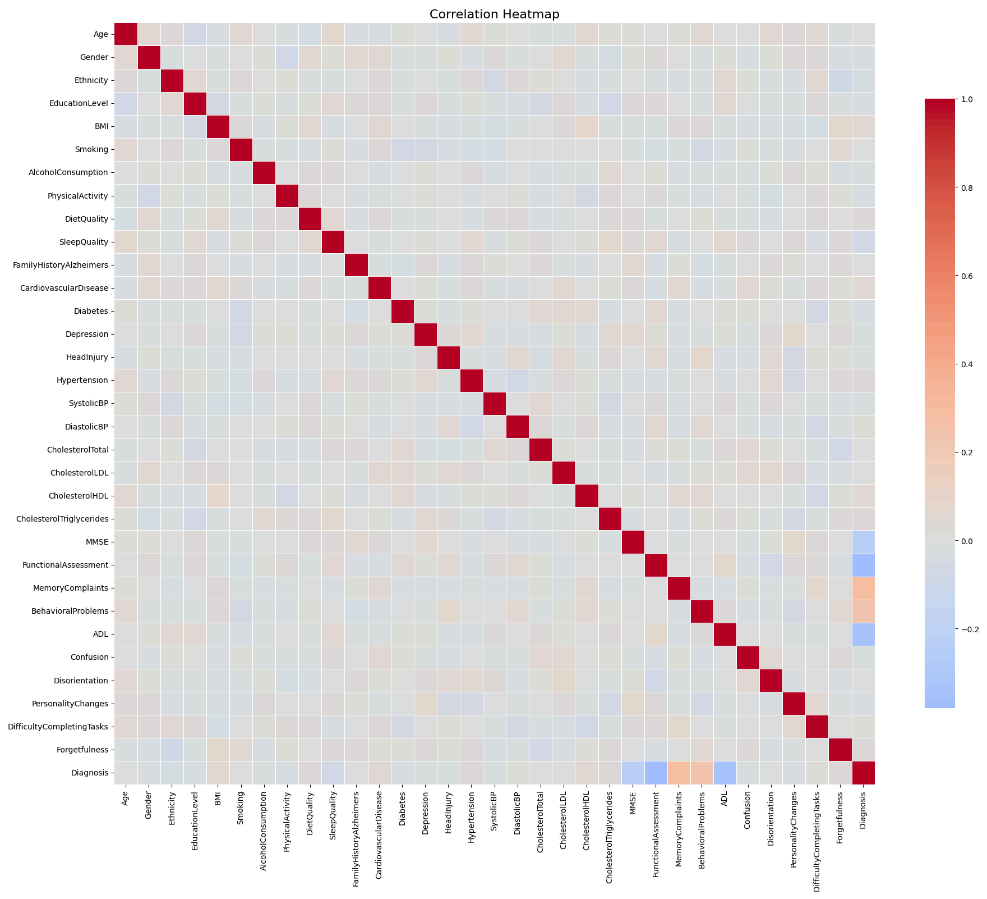


Figure 1: *A correlation heatmap (matrix) of all features in the training set, including the target (Diagnosis).*

To gain a deeper understanding of these 5 features, first, we looked at two distributions of “FunctionalAssessment” features based on “Diagnosis” status. Therefore, two PDFs have been plotted (Figure 2). By examining the graphs, people with benign status tend to have a standardized functional assessment score above 0, while people with malignant status tend to have a score below 0. This indicates that a DT split at 0 might make a good separation of the classes; however, the overlap between the distributions indicates that the split may not perfectly distinguish the classes, potentially reducing the effectiveness of DTs. We also plotted “MMSE” graphs with the same method (Figure 3). From visualizing the graphs, the standardized MMSE score below 1.0 does not give any valuable information; however, almost every person with a score above 1.0 is benign, suggesting that a DT split at 1.0 would result in a near-perfect separation of the classes which strengthened the compatibility of DT to the problem. From these three graphs, we expected that a DT would be a suitable choice of model. In addition, we have concluded that these 5 features are important in training our model. This is our first approach to feature selection. Next, we will need to visualize outliers in these features. Therefore, we consider three features which are numerical values: Functional Assessment, ADL, and MMSE. We plotted three box plots and found that the only feature with outliers is ADL. In this case, we will only show the box plot of ADL (Figure 4). We found that 9 data points are outliers. Since DTs are robust to outliers and the amount of outliers is fairly small, we have decided to not eliminate them as we might miss important information from those data points.

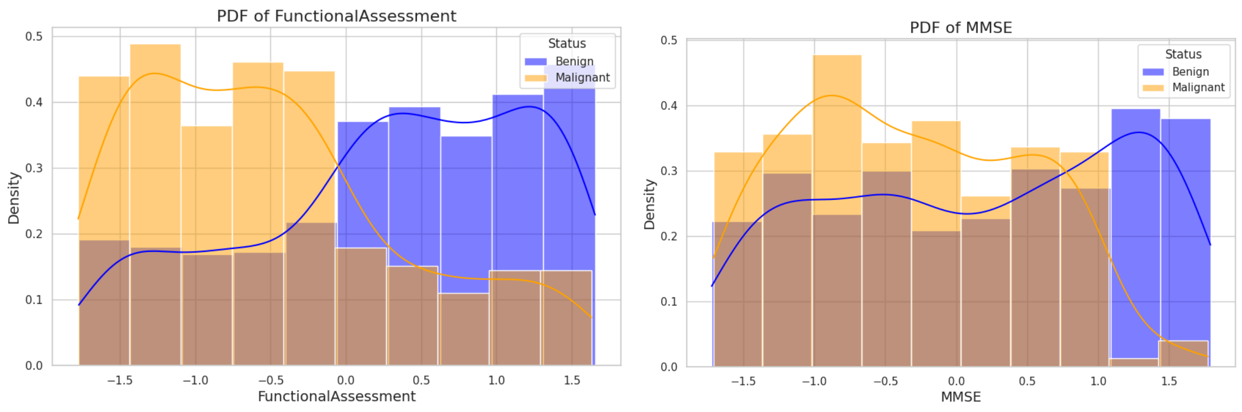


Figure 2 & 3: *Probability density functions of (left) standardized Functional Assessment (right) standardized MMSE based on Diagnosis status.*

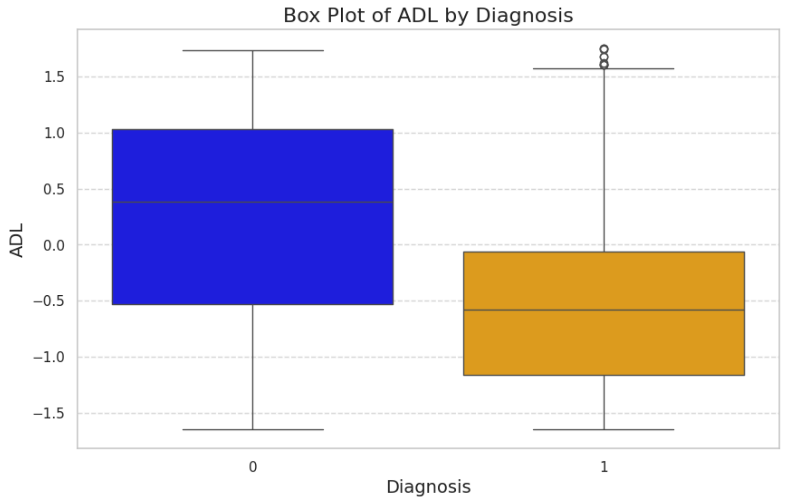


Figure 4: *A box plot of ADL feature distributions between the benign(0) and malignant(1) AD diagnosis. In agreement with the domain knowledge, ADL has a lower standardized median value among individuals affected by AD than among those unaffected.*

The second approach to our feature selection is based on random forest(RF) feature importances. We used RF which is more robust to high dimensional features to fit the dataset with all the columns and plotted its feature importances (Figure 5).

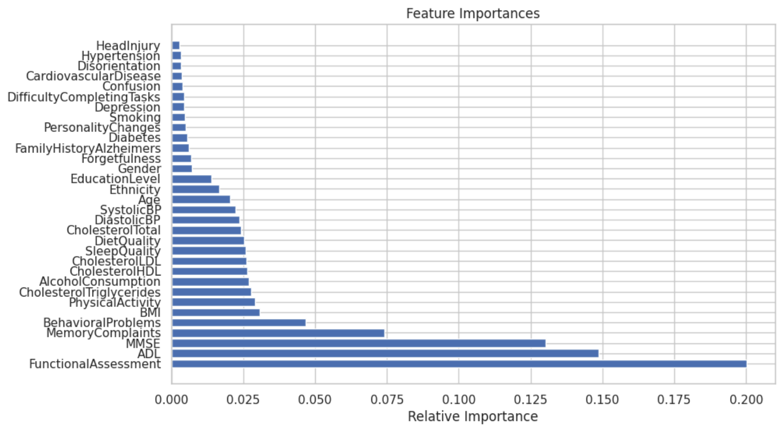


Figure 5: *A bar plot of feature importance from a random forest(RF) model.*

The features' importance matched our analysis having the five aforementioned 5 features as the most important features. Then, we selected the 30 most important features to fit boosted tree models.

2.3 Model

All of the following models are trained on the training set and evaluated on the validation set, which we have obtained from splitting “train.csv”. Then, we evaluate the best model on the test set from Kaggle. To solve data imbalance by generating synthetic positive AD samples, we have included the Synthetic Minority Oversampling Technique (SMOTE) in our training set which has resulted in better validation accuracy. Lastly, each model can take different approaches to feature selection when fitting.

Logistic Regression

Logistic Regression (LR) is a model which uses the sigmoid function to map input features to the probabilities of being in a class. It assumes a linear relationship between the input features and the log-odds (logit) of the target class probabilities, making it best suited for data that is linearly separable. Since LR is fairly simple to implement, we have decided to test LR, even though the EDA part suggested that our data would be most suited for decision trees(DTs). This model would serve as our baseline where we expected that DTs would achieve better performance, and it would also be beneficial in ensembles which require a diversity of multiple models. We fitted the model using the top 5 features on our resampled training set with L2 ridge regularization and liblinear solver. As a result, it achieved a validation accuracy of 0.8 and a validation recall of 0.8. In our approach, LR and a single DT classifier are used as predictors of the top 5 features with high correlation to the diagnosis and low intercorrelation.

Decision Tree

Decision Tree(DT) is a tree-like structure that splits data into subsets based on feature values at each node, with the goal to create subsets that ideally contain only one class of the outputs for prediction. We have decided to use this model with high expectations as the results from the EDA part have suggested the high compatibility of DTs to our data. As a single DT tends to overfit, we used the 5 most important features of the resampled training set similar to LR in fitting the model to reduce its flexibility. Then, we found that the DT is still overfitting, so we limited the max depth to 5 and the min samples split to 2. Finally, it achieved a validation accuracy of 0.927 and a validation recall of 0.927, with minimal overfitting. This highlighted the effectiveness of DTs in dealing with patterns from our data, leading us to focus on tree-based algorithms for the remaining models. In addition, The DT classifier mimics the doctor's decision leveraging straightforward decision boundaries of the top 5.

Random Forest

Random Forest(RF) is an ensemble model that trains multiple DTs, each on a bootstrap sample of the training dataset, while applying feature bagging (RF selection) at each split to enhance diversity among the trees. The goal of RF is to decrease overfitting by averaging over multiple DTs and achieve more accurate prediction while trading off interpretation compared to a single DT. Since our goal is to improve our ranking on the Kaggle leaderboard, and both objectives align, we decided to use RF to enhance prediction performance. As RF are robust to high dimensional data, we trained the model using all the features from our resampled training set. We used a grid search of 5 fold with an extensive parameters grid for our tuning method. The grids include “n_estimators”, “max_depth”, “min_samples_split”, “min_samples_leaf”, and “max_features”. Finally, it achieved a validation accuracy of 0.92 and a validation recall of 0.92. We also used feature importances obtained from this model as our feature selection in tree boosting.

Tree Boosting (Final Model)

Tree Boosting is an ensemble method that trains multiple weak DTs sequentially, where each tree is trained to correct the errors made by the previous trees. To clarify, each tree is trained on the negative gradients of the loss function of previous trees, allowing the ensemble to iteratively improve its performance as more trees are added. Similar to RF, Tree Boosting improves model performance, but it comes at the cost of interpretability. Bentéjac et al. (2020) suggested that Tree Boosting models are able to outperform RFs in multiple tasks with the requirement of proper hyperparameter tuning. Therefore, we have selected tree-boosting models to improve our prediction performance even further. We tried three variations of gradient-boosted trees from different libraries: XGBoost, SKlearn, and CatBoost. Each variation was fitted using the 30 most important features obtained from RF on the resampled training set. We utilized grid search for all three models with parameters grid including “n_estimators”, “learning_rate”, and “max_depth”. We used 5-fold cross-validation and have experimented with multiple scoring metrics. As a result, precision scoring has given the best result based on validation accuracy. From all of these, CatBoost has outperformed the other two models and has achieved a validation accuracy of 0.947 and a validation recall of 0.947. Comparing the results of all models, CatBoost had achieved the best validation accuracy of 0.947; therefore, we refitted CatBoost on the whole dataset with resampling to obtain our final model which is used to predict labels in the test set.

Ensembling

The ensemble approach is a method to combine the prediction of multiple models together with the focus on obtaining even better predictions. It consists of multiple methods such as averaging predictions, model stacking, soft voting, and hard voting. As we have struggled to find any other models that could potentially boost our prediction performance, we have decided to combine the discussed models together to try to improve our prediction. For our approach, the best model from each of the models we have tried has been selected for the weighted averaging method. To clarify, we obtain the prediction of each model before multiplying weighting factors that sum up to 1. Finally, we sum the weighted prediction together. The weights assigned were 0.05 for logistic regression, 0.05 for decision tree, 0.2 for random forest, 0.2 for XGBoost, 0.2 for SKlearn gradient boosted tree, and 0.3 for Catboost, and we have obtained a validation accuracy of 0.9435 and a validation recall of 0.9435.

2.4 Evaluation

To determine the evaluation metrics, we looked at the class distribution of the label “Diagnosis” (Figure 6). From this, it is clear that the data is imbalanced(benign/negative~65%,malignant/positive~35%). Therefore, the accuracy metric is inappropriate as the metric will largely be based on the major “benign” class. In a realistic situation, the most appropriate metric is recall which emphasizes the ability of a model to correctly identify all relevant instances of the positive class, particularly focusing on minimizing false negatives. This is crucial as false negatives in AD's diagnosis can lead to patients missing early medical attention, which may result in much worse symptoms in the later stages of the disease. However, as the Kaggle leaderboard is based on prediction accuracy, we used accuracy as our primary evaluation metric with recall as a secondary metric. From this, precision has been used as a scoring metric in our grid search as it has provided us with the best result based on accuracy.

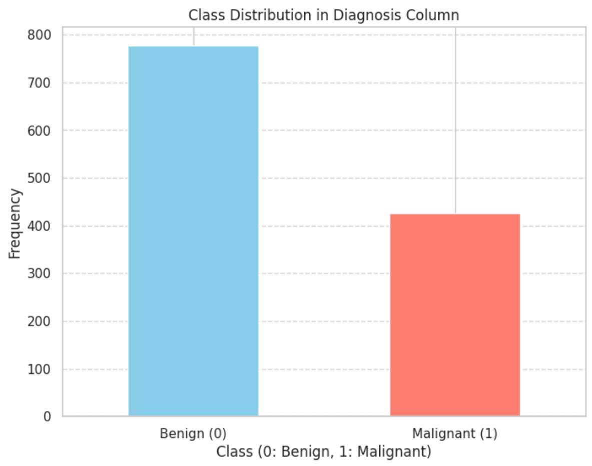


Figure 6: *A bar graph showing the distribution of diagnosis groups in the dataset*

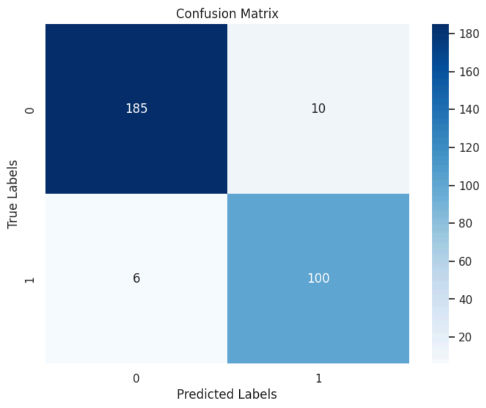
3 Results and Conclusions

In this study, we explored various types of models for classifying Alzheimer’s disease in patients, eventually selecting the CatBoost model due to its superior performance (Table 1). Through training with parameters and features selection, the CatBoost classifier achieved an accuracy of 0.947 on the validation set and a satisfied AUC graph as shown in Figure 8 reflecting its effectiveness in our classification problem. In addition, the refitted CatBoost model achieved a final accuracy of 0.92628, placing us in 18th place on the Kaggle leaderboard. This result is consistent with our EDA, which is to choose using a DT-based model.

Model	LR	DT	RF	XGBoost	SKlearn	CatBoost	Ensemble
Validation Accuracy	0.8	0.927	0.92	0.94	0.927	0.947	0.9435
Validation Recall	0.8	0.927	0.92	0.94	0.927	0.947	0.9435

Table 1: *Accuracy and recall on the validation set from train.csv*

Catboost was resilient to noise and overfitting in our dataset due to its efficient handling of categorical features and ordered boosting mechanism. Ordered boosting ensures that while training each tree, only known training data is used to avoid leakage of future training data and reduce the possibility of overfitting. This mechanism allowed us to train on 30 different features, achieving high validation accuracy despite the large number of features. In addition to the effectiveness of ordered boosting, SMOTE resampling helped prevent the model from overfitting to the majority class, as evidenced by the confusion matrix (Figure 7), which shows relatively balanced misclassification rates for both classes.



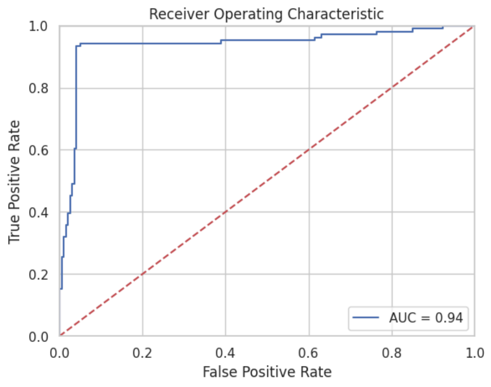
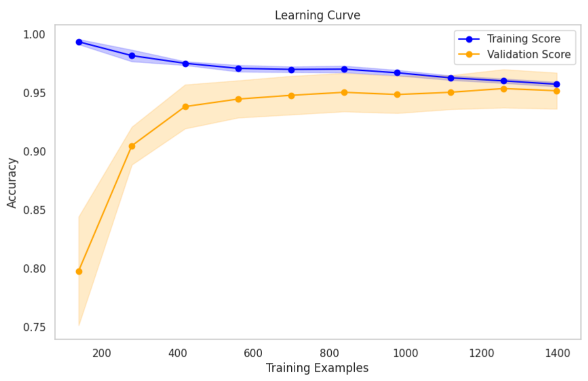


Figure 7&8: *The confusion matrix and AUC graph given by the final prediction of the validation set from the Catboost model.*

The learning curve of the model (Figure 9) demonstrates that the effect of overfitting decreases as the number of data increases. The training accuracy remains high (close to 0.97), reflecting the model's ability to fit the training data well, and the validation accuracy increases steadily with more training examples, eventually stabilizing at a high value close to the training score. The early stages of the graph indicate overfitting as the training accuracy has a value close to 1, while the validation accuracy is very low. This effect decreases as the gap between validation and training accuracies becomes smaller.
As shown in Figure 10, Grid Search gives us the best parameter choice iteration 6th about 4% more test scores than the worst parameter choice iteration 19th, which is a reasonable improvement due to the model's adaptive function to specific data.



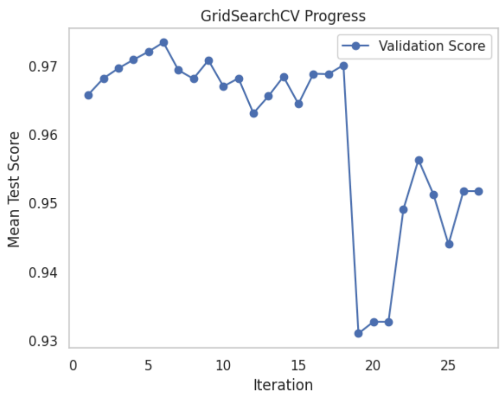


Figure 9&10: *Learning curve and A line chart showing each attempt of Grid Search and the test score for the final Catboost model*

Throughout this project, from a statistical perspective, we have gained a deeper understanding of AD. Our effective ML model will help to more accurately determine whether a patient is in the early stages of AD. We have identified features that have an important impact on disease prediction as shown in the EDA part. "FunctionalAssessment", "ADL", "MMSE", "MemoryComplaints", and "BehavioralProblems", etc are clinical data to be considered in order of importance. The discovery of the importance of these features provides a new perspective on the early warning mechanism of AD, making early diagnosis possible and improving the speed and accuracy of diagnosis.

4 Discussion

Although CatBoost is very efficient in handling categorical features and training, the hyperparameter optimization used by GridSearchCV is computationally expensive. The datasets we are dealing with so far are relatively simple. However, when the data has more features or the sample size is larger, the training time will be greatly increased, and it may also cause greater pressure on computing resources.
Future improvement work could focus on exploring more hyperparameter search methods, distributed and parallel computing methods, and tree compressing methods like TreeSHAP pruning.

During our exploratory data analysis, we also found that the current dataset only contains about five key features, which constrains the model from achieving a higher level of performance. Further improvements could be focused on introducing more complete and diverse features, such as MRI or CT scan pictures. More detailed information exploration can provide more data support for the improvement of the model. Moreover, testing other models‘ approaches might lead to a better performance as well. For example, neural networks could be effective if we can expand the dataset, and convolutional neural networks (CNNs) might capture more complex relationships in the spatial information from MRI scans. With richer data input and more advanced modelling techniques, we expect to provide more accurate and reliable predictions.

5 References

[1] An, J. (2022). Using CatBoost and Other Supervised Machine Learning Algorithms to Predict Alzheimer’s Disease. doi:https://doi.org/10.1109/icmla55696.2022.00265.

[2] Bentéjac, C., Csörgő, A. and Martínez-Muñoz, G. (2020). A comparative analysis of gradient boosting algorithms. *Artificial Intelligence Review*, 54(3). doi:https://doi.org/10.1007/s10462-020-09896-5. https://arxiv.org/pdf/1911.01914

[3] Geerlings, M.I., Jonker, C., Bouter, L.M., Adèr, H.J. and Schmand, B. (1999). Association Between Memory Complaints and Incident Alzheimer’s Disease in Elderly People With Normal Baseline Cognition. *American Journal of Psychiatry*, 156(4), pp.531–537. doi:https://doi.org/10.1176/ajp.156.4.531. https://pubmed.ncbi.nlm.nih.gov/10200730/

[4] Grammenos, G., Vrahatis, A.G., Vlamos, P., Palejev, D. and Exarchos, T. (2024). Predicting the Conversion from Mild Cognitive Impairment to Alzheimer’s Disease Using an Explainable AI Approach. *Information*, 15(5), p.249. doi:https://doi.org/10.3390/info15050249.

[5] Alzheimer's Association (2023a). 2023 Alzheimer’s Disease Facts and Figures. *Alzheimer’s & Dementia*, 19(4), pp.1598–1695. doi:https://doi.org/10.1002/alz.13016.

[6] Solomon, T.M., deBros, G.B., Budson, A.E., Mirkovic, N., Murphy, C.A. and Solomon, P.R. (2014). Correlational Analysis of 5 Commonly Used Measures of Cognitive Functioning and Mental Status. *American Journal of Alzheimer’s Disease & Other Dementiasr*, 29(8), pp.718–722. doi:https://doi.org/10.1177/1533317514534761.

[7] Kaggle Contributors. (2024) Alzheimer’s Disease Dataset. Retrieved from https://www.kaggle.com/competitions/classification-of-the-alzheimers-disease

[8] James, G., Witten, D., Hastie, T. and Tibshirani, R. (2013). *An introduction to statistical learning : with applications in R*. [online] Springer. Available at: https://static1.squarespace.com/static/5ff2adbe3fe4fe33db902812/t/6009dd9fa7bc363aa822d2c7/1611259312432/ISLR+Seventh+Printing.pdf.

Appendix

6 Code

https://colab.research.google.com/drive/1tI9v8XbEUeOnEzmwntv-U6vkxVf9bpV_?usp=sharing

```
!pip install ISLP
!pip install catboost
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.pyplot import subplots
import statsmodels.api as sm
from ISLP import load_data
from ISLP.models import (ModelSpec as MS,
summarize)

from ISLP import confusion_table
from ISLP.models import contrast
from sklearn.discriminant_analysis import \
(LinearDiscriminantAnalysis as LDA,
QuadraticDiscriminantAnalysis as QDA)
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from itertools import combinations

import seaborn as sns
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, f1_score, recall_score,
balanced_accuracy_score,classification_report, confusion_matrix
from sklearn.model_selection import GridSearchCV
from imblearn.over_sampling import SMOTE
from sklearn.ensemble import RandomForestClassifier
import xgboost as xgb
from sklearn.linear_model import LogisticRegression


df = pd.read_csv("train.csv")
df.head(5)


#Checking for NULL values
df.isna().sum()
```

```python
#Checking for duplicates
duplicates = df[df[list(df.columns)].duplicated()].index
duplicates

def reset_df():
    # 0.8-0.2 Train-val split
    df_train, df_val = train_test_split(df, test_size=0.2, random_state=5,
stratify=df['Diagnosis'])

    train_diagnosis = df_train['Diagnosis'].reset_index(drop = True)
    val_diagnosis = df_val['Diagnosis'].reset_index(drop = True)

    #Dropping Irrelevant columns
    df_train = df_train.drop(['PatientID', 'DoctorInCharge','Diagnosis'], axis=1)
    df_val = df_val.drop(['PatientID', 'DoctorInCharge','Diagnosis'], axis=1)

    #Standardizing every column except diagnosis
    scaler = StandardScaler()

    df_train = pd.DataFrame(scaler.fit_transform(df_train),
columns=df_train.columns)
    df_val = pd.DataFrame(scaler.transform(df_val), columns = df_val.columns)

    #Adding diagnosis column which is not standardized
    df_train['Diagnosis'] = train_diagnosis
    df_val['Diagnosis'] = val_diagnosis
    return df_train, df_val, scaler
df_train, df_val, scaler = reset_df()

#Plotting Correlation Matrix
correlation_matrix = df_train.corr()
plt.figure(figsize=(20, 16))
sns.heatmap(correlation_matrix, annot=False, cmap='coolwarm', center=0,
            square=True, linewidths=0.5, cbar_kws={"shrink": .8})
plt.title('Correlation Heatmap', fontsize=16)
plt.tight_layout()
plt.show()
```

```python
def top_correlated_features(data, target_feature, top_n=10):
    # Identifies the top correlated features to a given target feature.

    # Calculate the correlation matrix
    correlation_matrix = data.corr()

    # Extract correlation values for the target feature
    target_correlation =
correlation_matrix[target_feature].drop(labels=[target_feature])

    # Get the top N correlated features (absolute value to account for both
positive and negative correlations)
    top_correlations =
target_correlation.abs().sort_values(ascending=False).head(top_n)

    # Return the correlation values with original signs
    return target_correlation[top_correlations.index]
top_correlated_features(df_train,"BehavioralProblems")

def plot_pdf(df, col):
    #Plot two pdf: pdf of specific column given diagnosis = 0, pdf of specific
column given diagnosis = 1
    sns.set(style="whitegrid")

    # Plot the PDF using Seaborn's kdeplot
    plt.figure(figsize=(10, 6))
    sns.histplot(df[df['Diagnosis'] == 0][col], kde=True, color='blue',
label='Benign', stat="density", bins=10, alpha=0.5)
    sns.histplot(df[df['Diagnosis'] == 1][col], kde=True, color='orange',
label='Malignant', stat="density", bins=10, alpha=0.5)

    plt.title(f"PDF of {col}", fontsize=16)
    plt.xlabel(f"{col}", fontsize=14)
    plt.ylabel("Density", fontsize=14)
    plt.legend(title="Status")
    plt.grid(True)

    plt.show()
plot_pdf(df_train,"FunctionalAssessment")

plot_pdf(df_train,'MMSE')

def plot_box(df,col):
    #Plotting box plot of specific column for diagnosis = 0, diagnosis = 1

    plt.figure(figsize=(10, 6))
    sns.boxplot(data=df, x='Diagnosis', y=col, palette={'0': 'blue', '1':
'orange'})

    plt.title(f'Box Plot of {col} by Diagnosis', fontsize=16)
    plt.xlabel('Diagnosis', fontsize=14)
    plt.ylabel(col, fontsize=14)

    plt.grid(axis='y', linestyle='--', alpha=0.7)
    plt.show()
plot_box(df_train,"FunctionalAssessment")
plot_box(df_train,"ADL")


#Finding upper bound for group 1
grouped = df_train.groupby('Diagnosis')['ADL']

# For Diagnosis = 1 (orange box)
Q1 = grouped.quantile(0.25).loc[1]
Q3 = grouped.quantile(0.75).loc[1]
IQR = Q3 - Q1
upper_bound = Q3 + 1.5 * IQR

# Print results for Diagnosis = 1
print("For Diagnosis = 1:")
print("Q1:", Q1)
print("Q3:", Q3)
print("IQR:", IQR)
print("Upper Bound:", upper_bound)

len(df_train[(df_train['ADL'] > upper_bound) & (df_train['Diagnosis'] == 1)])

plot_box(df_train,"MMSE")

plot_box(df_train,"SleepQuality")

class_counts = df_train['Diagnosis'].value_counts()

# Print the distribution
print("Class Distribution:")
print(class_counts)

# Plot the distribution
plt.figure(figsize=(8, 6))
class_counts.plot(kind='bar', color=['skyblue', 'salmon'])
plt.title('Class Distribution in Diagnosis Column')
plt.xlabel('Class (0: Benign, 1: Malignant)')
plt.ylabel('Frequency')
plt.xticks([0, 1], ['Benign (0)', 'Malignant (1)'], rotation=0)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()

def evaluate(model, X, y, class_report = True):
    y_pred = model.predict(X)
    accuracy = accuracy_score(y,y_pred)
    f1 = f1_score(y,y_pred, average='weighted')
    sensitivity = recall_score(y,y_pred,average='weighted')
    balanced_accuracy = balanced_accuracy_score(y,y_pred)
    if class_report:
        print(classification_report(y, y_pred,
target_names=["Benign","Malignant"]))
    return accuracy, f1, sensitivity, balanced_accuracy


def evaluate_model(model, X_train, y_train, X_val, y_val, class_report = True):
    train_accuracy, train_f1, train_sensitivity, train_balanced_accuracy = 
evaluate(model, X_train, y_train, class_report)
    validation_accuracy, validation_f1, validation_sensitivity, 
validation_balanced_accuracy = evaluate(model, X_val, y_val, class_report)
    print(f"Training Metrics: Accuracy = {train_accuracy}, F1 score = {train_f1}, 
Sensitivity = {train_sensitivity}")
    print(f"Validation Metrics: Accuracy = {validation_accuracy}, F1 score = 
{validation_f1}, Sensitivity = {validation_sensitivity}")

def prepare_X_y(X_col, resample = True):
    #prepare X, y given arbitrary training features with SMOTE resampling
    X_train = df_train[X_col]
    y_train = df_train['Diagnosis']

    X_val = df_val[X_col]
    y_val = df_val['Diagnosis']

    #Apply SMOTE resampling
    if resample == True:
        smote = SMOTE(random_state=0)
        X_train, y_train = smote.fit_resample(X_train, y_train)
    return X_train, y_train, X_val, y_val


#Try using MMSE, FunctionalAssessment, MemoryComplaints, BehavioralProblems, ADL 
for fitting (5 most important features)
X_train, y_train, X_val, y_val = prepare_X_y(['MMSE', 'FunctionalAssessment', 
'MemoryComplaints', 'BehavioralProblems', 'ADL',])
lr_model = LogisticRegression(
    penalty='l2',  # L2 regularization (ridge)
    C=1.0,
    solver='liblinear',  # Good for small datasets
    random_state=42,
    max_iter=1000,
)

# Fit the model
lr_model.fit(X_train, y_train)
evaluate_model(lr_model, X_train, y_train, X_val, y_val)

#Decision Tree
clf = DecisionTreeClassifier(random_state=0, max_depth = 5, min_samples_split = 
2)
clf.fit(X_train, y_train)
```

```python
evaluate_model(clf, X_train, y_train, X_val, y_val)

import shap

# Initialize SHAP explainer
explainer = shap.TreeExplainer(clf)

# Get SHAP values for the entire dataset
shap_values = explainer.shap_values(X_val)

# Visualize SHAP values
shap.summary_plot(shap_values[:,:,1], X_val)  # For binary classification (class
1)

#Random Forest using all features & Grid search
features = [col for col in df_train.columns if col != 'Diagnosis']
X_train, y_train, X_val, y_val = prepare_X_y(features)

param_grid = {
    'n_estimators': [100, 200],
    'max_depth': [None, 10],
    'min_samples_split': [2, 5],
    'min_samples_leaf': [1, 2, 3],
    'max_features': ['log2','sqrt'],
    'bootstrap': [True]
}

rf = RandomForestClassifier(random_state=42)
grid_search = GridSearchCV(estimator=rf, param_grid=param_grid, cv=5,
scoring='precision', verbose=1) #Accuracy
grid_search.fit(X_train, y_train)


print("Best Parameters:", grid_search.best_params_)
print("Best Accuracy:", grid_search.best_score_)
best_rf = grid_search.best_estimator_


#Feature importances from our Random Forest
importances = best_rf.feature_importances_

feature_names = X_train.columns

indices = np.argsort(importances)[::-1]

# Plot the feature importances
plt.figure(figsize=(10, 6))
plt.title("Feature Importances")
plt.barh(range(len(importances)), importances[indices], align="center")
plt.yticks(range(len(importances)), np.array(feature_names)[indices])
plt.xlabel("Relative Importance")
plt.show()

validation_accuracy, validation_f1, validation_sensitivity,
validation_balanced_accuracy = evaluate(best_rf, X_val, y_val)
print(f"Validation Metrics: Accuracy = {validation_accuracy}, F1 score =
{validation_f1}, Sensitivity = {validation_sensitivity}")

#Fitting XGBoost on 30 most important features
X_train, y_train, X_val, y_val =
prepare_X_y(np.array(feature_names)[indices][:30])

xgb_clf = xgb.XGBClassifier(
    objective='binary:logistic',
    random_state=42,
    n_jobs=-1,  # Use all available cores
)

xgb_clf.fit(X_train,y_train)
evaluate_model(xgb_clf,X_train, y_train, X_val, y_val)


#XGBoost with grid search
param_grid = {
              'n_estimators': [50, 100, 200],
              'learning_rate': [0.01, 0.1, 1],
              'max_depth': [3, 5, 7],
              }
# Create the XGBoost classifier
xgb_clf = xgb.XGBClassifier(
    objective='binary:logistic',
    random_state=42,
    n_jobs=-1  # Use all available cores
)

grid_search = GridSearchCV(estimator=xgb_clf, param_grid=param_grid, cv=5,
scoring='precision', verbose=1)

grid_search.fit(X_train, y_train)


print("Best Parameters:", grid_search.best_params_)
print("Best Accuracy:", grid_search.best_score_)
best_xgb = grid_search.best_estimator_


validation_accuracy, validation_f1, validation_sensitivity,
validation_balanced_accuracy = evaluate(best_xgb, X_val, y_val)
```

```
print(f"Validation Metrics: Accuracy = {validation_accuracy}, F1 score =
{validation_f1}, Sensitivity = {validation_sensitivity}")


#SKlearn gradient boosting + Grid search
from sklearn.ensemble import GradientBoostingClassifier
model = GradientBoostingClassifier(random_state = 0)

grid_search = GridSearchCV(model, param_grid, cv = 5, scoring = 'precision')
grid_search.fit(X_train,y_train)
best_gb = grid_search.best_estimator_

print("Best Parameters:", grid_search.best_params_)
print("Best Accuracy:", grid_search.best_score_)

validation_accuracy, validation_f1, validation_sensitivity,
validation_balanced_accuracy = evaluate(best_gb, X_val, y_val)
print(f"Validation Metrics: Accuracy = {validation_accuracy}, F1 score =
{validation_f1}, Sensitivity = {validation_sensitivity}")

#CatBoost + Gridsearch
from catboost import CatBoostClassifier
cbc = CatBoostClassifier(verbose=0)
grid_search = GridSearchCV(estimator=cbc, param_grid=param_grid, cv=5,
scoring='precision', verbose=1)

grid_search.fit(X_train,y_train)
best_cbc = grid_search.best_estimator_

# taking results
results = grid_search.cv_results_

# checking result tables
results_df = pd.DataFrame(results)
print(results_df[['param_learning_rate', 'param_max_depth','param_n_estimators',
'mean_test_score']])

# compute score
scores = results['mean_test_score']
iterations = range(1, len(scores) + 1)

# line graph
plt.plot(iterations, scores, marker='o', label='Validation Score')
plt.xlabel('Iteration')
plt.ylabel('Mean Test Score')
plt.title('GridSearchCV Progress')
plt.grid()
plt.legend()
plt.show()
```

```
from sklearn.model_selection import learning_curve
# define model
model = CatBoostClassifier(**best_cbc.get_params())

# calculating learning curve
train_sizes, train_scores, test_scores = learning_curve(
    model, X_train, y_train, cv=10, scoring='accuracy', n_jobs=-1,
train_sizes=np.linspace(0.1, 1.0, 10)
)

train_mean = np.mean(train_scores, axis=1)
train_std = np.std(train_scores, axis=1)
test_mean = np.mean(test_scores, axis=1)
test_std = np.std(test_scores, axis=1)

# plot
plt.figure(figsize=(10, 6))
plt.plot(train_sizes, train_mean, 'o-', color='blue', label='Training Score')
plt.plot(train_sizes, test_mean, 'o-', color='orange', label='Validation Score')

# std area
plt.fill_between(train_sizes, train_mean - train_std, train_mean + train_std,
alpha=0.2, color='blue')
plt.fill_between(train_sizes, test_mean - test_std, test_mean + test_std,
alpha=0.2, color='orange')


plt.title('Learning Curve')
plt.xlabel('Training Examples')
plt.ylabel('Accuracy')
plt.legend(loc='best')
plt.grid()
plt.show()

print("Best Parameters:", grid_search.best_params_)
print("Best Accuracy:", grid_search.best_score_)

validation_accuracy, validation_f1, validation_sensitivity,
validation_balanced_accuracy = evaluate(best_cbc, X_val, y_val)
print(f"Validation Metrics: Accuracy = {validation_accuracy}, F1 score =
{validation_f1}, Sensitivity = {validation_sensitivity}")

#adddded for confusion matrix
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt

y_true = y_val
```

```
y_pred = best_cbc.predict(X_val)

#compute
cm = confusion_matrix(y_true, y_pred)

#display
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix')
plt.show()

#AUC
import sklearn.metrics as metrics
probs = best_cbc.predict_proba(X_val)
preds = probs[:,1]
fpr, tpr, threshold = metrics.roc_curve(y_true, preds)
roc_auc = metrics.auc(fpr, tpr)

#method: plt
import matplotlib.pyplot as plt
plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()

#Refitting the model on whole data

#Top 30 features
features = list(X_train.columns)

diag = df['Diagnosis'].reset_index(drop = True)

#Drop Irrelevant Columns
df_scaled = df.drop(['PatientID', 'DoctorInCharge','Diagnosis'], axis=1)

#Standardizing every column except diagnosis
df_scaled = pd.DataFrame(scaler.transform(df_scaled), columns=df_scaled.columns)

#Adding unscaled diagnosis column back
df_scaled['Diagnosis'] = diag

#Get X,y
```

```
X = df_scaled[features]
y = df_scaled['Diagnosis']
smote = SMOTE(random_state=20)

#Resample
X, y = smote.fit_resample(X, y)

#Refit the model on whole data
best_model = CatBoostClassifier(**best_cbc.get_params())
best_model.fit(X, y)


X_train, y_train, X_val, y_val = prepare_X_y(['MMSE', 'FunctionalAssessment',
'MemoryComplaints', 'BehavioralProblems', 'ADL',])

#Logistic regression
lr_pred = lr_model.predict_proba(X_val)

#Decision Tree
dt_pred = clf.predict_proba(X_val)

#Random Forest
features = [col for col in df_train.columns if col != 'Diagnosis']
X_train, y_train, X_val, y_val = prepare_X_y(features)

rf_pred = best_rf.predict_proba(X_val)

#Tree Boosting
X_train, y_train, X_val, y_val =
prepare_X_y(np.array(feature_names)[indices][:30])
xgb_pred = best_xgb.predict_proba(X_val)
gb_pred = best_gb.predict_proba(X_val)
cbc_pred = best_cbc.predict_proba(X_val)

#Weighted Averaging
y_pred_prob = (0.05*lr_pred + 0.05*dt_pred + 0.2*rf_pred + 0.2*xgb_pred +
0.2*gb_pred + 0.3*cbc_pred)
y_pred = np.argmax(y_pred_prob, axis=1)
print(f"Accuracy: {accuracy_score(y_val,y_pred)}")
print(f"Recall: {recall_score(y_val,y_pred, average='weighted')}")

#model submission
df_bf = pd.read_csv('best_submission.csv')
df_test = pd.read_csv("test.csv")


def get_submission(model, features_used,scaler):
    df_test = pd.read_csv("test.csv")
    pat_ID = pd.DataFrame(df_test['PatientID'])
    df_test = df_test.drop(['PatientID', 'DoctorInCharge'], axis=1)
    df_test = pd.DataFrame(scaler.transform(df_test), columns=df_test.columns)

    X_t = df_test[features_used]
    y_test_pred = model.predict(X_t)
    pat_ID['Diagnosis'] = y_test_pred
    pat_ID.to_csv('submission.csv', index=False)

    accuracy, f1, sensitivity, _ = evaluate(model, X_t, df_bf['Diagnosis'])
    print(f"Comparing best result: Accuracy = {accuracy}, F1 score = {f1},
Sensitivity = {sensitivity}")

    print("Submission File Downloaded")

get_submission(best_model, list(X_train.columns), scaler)

%%shell
jupyter nbconvert --to html /content/STA314_Project.ipynb
```