Optimization Sprint Report

Team Adagard

Name	University	NIC
Nivakaran Shanmugabavan	SLIIT	200208902949

1. Data Exploration and Process Flow

The dataset provided is a variation of the NACC (National Alzheimer's Coordinating Center) dataset, containing over 200 mixed-type features (numeric, categorical) and NACC-specific missing value codes (e.g., 99, -4) . Our process flow was an iterative experiment designed to be robust, compliant, and prevent data leakage:

1. Rule-Based Filtering (Manual Selection): We first defined an "allow-list" of 58 hackathon-compliant, non-medical, self-reported features (e.g., NACCAGE, EDUC, CVHATT). A diagnostic check (Cell 5) confirmed that only 39 of these desired features were actually present in the provided dataset.
2. Leakage Prevention: A critical step was identifying INDEPEND as a data leakage variable. We preemptively removed it from our feature list (Cell 4) to ensure a true predictive model.
3. Data Cleaning: We standardized missingness by replacing NACC codes with np.nan and simplifying diagnosis codes to binary (functions in Cell 3, applied in Cells 4 & 6).
4. Feature Engineering: We created two new aggregate features, FE_COMORBIDITY_SCORE and FE_SLEEP_PSYCH_SCORE, by summing related health conditions (Cell 6).
5. Automated Feature Selection: We then used a RandomForestClassifier (tuned with GridSearchCV in Cell 12) to analyze all 41 features (39 original + 2 engineered) and rank them by importance. This model-based approach was chosen over simple correlation because it captures non-linear relationships and proved more accurate.
6. Exploratory Data Analysis (EDA): We visualized the data (Cells 8-11), which revealed the 70.5% vs. 29.5% class imbalance, confirming our use of stratify=y (Cell 12) and roc_auc (Cell 13).
7. Preprocessing Pipeline: We built a robust ColumnTransformer (Cell 10) to automatically handle imputation, scaling, and one-hot encoding.
8. Model Tournament: We conducted a comprehensive 10-model tournament (Cells 13, 14, 15).
9. Advanced Analysis & Selection: The notebook is designed to dynamically select the winning model from results_df (Cell 15 output) for final analysis. (Note: The subsequent analysis cells 17, 18, 22 were not successfully run in the provided file).
10. Finalization: The complete pipeline is designed to be retrained and saved as a .joblib file.

2. Feature Engineering

Our feature strategy was a multi-stage process, combining manual, rule-based filtering with the creation of new aggregate features and finally, an automated, model-based importance ranking.

Feature Selection (Manual)

- Initial Filtering: We first performed a manual review of the data dictionary and hackathon rules to create an "allow-list" of 58 potential non-medical, self-reported features.
- Data Diagnostic: A diagnostic check (Cell 5) confirmed that only 39 of these 58 desired features were actually present in the provided dataset.

- Feature Reduction (Leakage Prevention): Our most important reduction was the preemptive removal of INDEPEND (Cell 4). We identified this as a data leakage variable, as it measures a patient's current independence level (a symptom) rather than a risk factor.

Feature Creation (Cell 6)

- Hypothesis: We hypothesized that the cumulative health burden would be a more powerful predictor than any single condition.
- What Succeeded: We successfully created two new features by summing related binary columns:
    - FE_COMORBIDITY_SCORE: Sum of all 18 "Known Diagnoses" columns.
    - FE_SLEEP_PSYCH_SCORE: Sum of all 9 "Health History" columns.
- What Failed: The notebook produced warnings that FE_SMOKING_RISK and FE_FAMILY_HISTORY_SCORE could not be built because the underlying columns (like TOBACYRS and FAMDEM) were missing from the dataset. These columns were mentioned in the dataset dictionary pdf.

Feature Selection (Automated)

- Technique: To validate our new features and identify the most important predictors, we used a RandomForestClassifier (tuned with GridSearchCV in Cell 12) to rank all 41 features (39 original + 2 engineered).
- Justification: We chose this tree-based method over a simple correlation matrix because it is more powerful. A Random Forest can capture non-linear relationships and feature interactions (how features work together), which a correlation analysis would miss.
- Validation: This approach was highly successful. The analysis (Cell 12 output) confirmed our hypothesis by ranking our engineered FE_COMORBIDITY_SCORE as the #5 most important feature.
- Finalized Features: The raw 41 features were fed into our preprocessor (Cell 10). This pipeline automatically generated the final, wide dataset of 52 features (after one-hot encoding) for the models to train on.

3. Data Preprocessing

Our preprocessing was handled by a ColumnTransformer (Cell 13) to ensure no data leakage from the test set.

1. NACC Code Cleaning: Replaced all NACC-specific missing codes (e.g., 99, 999, -4) with np.nan (function defined in Cell 3, applied in Cell 4).
2. Health Column Conversion: Mapped diagnoses (0=Absent, 1=Active, 2=Inactive) to a simple binary (0=Absent, 1=Present) (function in Cell 3, applied in Cell 6).
3. Imputation & Scaling (Cell 10): We used three distinct strategies:
    - Numeric Features (38): Imputed using SimpleImputer(strategy='median'). We chose median as it is robust to outliers present in age and health data. This was followed by StandardScaler().
    - String Categorical Features (0): A transformer was built, but none were present in our final 39-feature set.

        - Numeric-Coded Categoricals (3): For SEX, MARISTAT, and RESIDENC, we used SimpleImputer(strategy='constant', fill_value=-999). This correctly treats missingness as its own distinct category, which is a powerful signal, rather than incorrectly imputing a common value like "Married".
4. Encoding (Cell 10): OneHotEncoder(handle_unknown='ignore') was applied to all 3 categorical features to convert them into a machine-readable format.
5. Train-Test Split (Cell 12): We used train_test_split(stratify=y). The stratify parameter was essential. It ensures that the 29.5% "At Risk" class (identified in Cell 8) was represented in the exact same proportion in both the training and test sets, which is critical for evaluating an imbalanced dataset.

4. Model Building

We conducted a 10-model tournament to ensure we found the best possible architecture. This was done in three stages:

1. GridSearchCV Tournament (Cell 13): We used GridSearchCV with 4-fold cross-validation (cv=4) and scoring='roc_auc' to tune and compare eight standard classifiers.
2. Meta-Ensemble (Cell 14): We trained a StackingClassifier using the best-tuned models from the previous step as base estimators.
3. Deep Learning (Cell 15): We trained a Keras-based Sequential (MLP) model as a deep learning challenger.

Models Trained (Cells 13, 14, 15):

- Linear: LogisticRegression (with 'liblinear' solver and L1/L2 penalty tuning).
- Ensemble (Bagging): RandomForestClassifier, ExtraTreesClassifier.
- Ensemble (Boosting): GradientBoostingClassifier, AdaBoostClassifier, XGBClassifier, LGBMClassifier, CatBoostClassifier.
- Meta-Ensemble (Cell 14): A StackingClassifier using the best-tuned LGBM, RF, and XGB models as base estimators and LogisticRegression as the final_estimator.
- Deep Learning (Cell 15): A Keras-based Sequential (MLP) model, incorporating Dropout layers and class_weight balancing, was also trained for experimental comparison. We did not pursue extensive optimization of this neural network, as traditional machine learning ensembles (like the tree-based models) are often more effective and generalize better on structured, tabular datasets of this size.

Class Imbalance Handling: For all models that support it (e.g., LGBM, RF, CatBoost, LogisticRegression, NeuralNetwork), we used their built-in class weight balancing parameters (e.g., class_weight='balanced') (Cells 17, 19). For XGBoost, we used scale_pos_weight=2.39 (the ratio of 70.5/29.5) (Cell 17).

Hyperparameter Tuning (Cell 17):

- We used GridSearchCV with 4-fold cross-validation (cv=4), not 5-fold as stated in the PDF report.
- Justification: This is a robust method to find the optimal hyperparameters for each model, ensuring we compare the "best version" of each against the others . The primary scoring metric was roc_auc..

5. Model Evaluation

- **Evaluation Metrics:**
    1. ROC_AUC (Primary): This was the primary metric for model comparison. It measures the model's ability to distinguish between classes, which is crucial for the 70.5% vs. 29.5% imbalanced dataset where accuracy would be misleading.
    2. F1-Score (Tuned): We did not use the default 0.5 threshold. For each model, the notebook dynamically calculated the decision threshold that maximized the F1-Score on the training data using precision_recall_curve to avoid data leakage (Cell 13).
    3. PR-AUC: Average Precision (Precision-Recall AUC) was also recorded.

- **Comparison of Each Model**: Our 10-model tournament, which included standard classifiers, a Stacking Ensemble, and a Neural Network, produced the following results on the unseen test set (Cells 13, 14, 15).

Model	**AUC**	**PR-AUC**	**F1 Score**	**Best Threshold**	**Time (s)**
Stacking_Ensemble	**0.8234**	**0.7102**	**0.6176**	**0.5814**	**1274.7632**
Random Forest	0.8166	0.6956	0.6163	0.4815	4003.5593
Extra Trees	0.8065	0.6863	0.6014	0.5140	5514.6733
XGBoost	0.7856	0.6563	0.5849	0.5174	388.6573
LightGBM	0.7846	0.6551	0.5854	0.5153	925.6813
GradientBoosting	0.7762	0.6435	0.5770	0.2892	2529.5125

CatBoost	0.7735	0.6379	0.5739	0.4934	429.2697
NeuralNetwork_MLP	0.7602	0.6220	0.5595	0.5000	N/A (See cell 15)
AdaBoost	0.7519	0.6054	0.5550	0.4478	639.7807
Logistic Regression	0.7420	0.5937	0.5445	0.4692	306.5772

(Source: Notebook outputs from Cells 13, 14, and 15)

- Final Model:
  - Dynamic Selection: The notebook is programmed to dynamically select the winner. The results_df (printed in Cell 15) is automatically sorted by "AUC" in descending order.
  - Justification: The winning model from the notebook's 10-model tournament is the Stacking_Ensemble, which achieved the highest Test AUC of 0.8234. All subsequent analysis cells (17, 18, 20) are programmed to use this winning model, ensuring the final report is based on the best-performing pipeline.

6. Explainability & Model Interpretability

- Explainability Technique: We used the feature_importances_ attribute, first in a preliminary analysis (Cell 15) and again on our dynamically-selected winning model (Cell 23) to identify the key drivers of its predictions.
- Insights from Explainability (Cell 15, 23):
  1. Top Drivers: The analyses consistently identified NACCBMI (Body Mass Index), NACCAGE (Age), and EDUC (Education) as the top predictors.
  2. Engineered Feature Success: Our engineered feature, FE_COMORBIDITY_SCORE, ranked as the #5 predictor (Cell 15), proving our hypothesis that cumulative health burden is a major predictor.
  3. Key Risk Factors: Other high-importance features included DEP2YRS (Depression), INCONTF (Fecal Incontinence), and INCONTU (Urinary Incontinence).
- Advanced Error & Threshold Analysis:
  1. Threshold Tuning (Cell 27): We analyzed the F1-Score, Precision, and Recall at the dynamically-found optimal threshold (e.g., 0.54, as mentioned in the old report, but calculated live in Cell 17). This threshold provides the best balance between minimizing "False Alarms" and "Dangerous Misses."
  2. Error Analysis (Cell 27): We isolated and profiled "False Negatives" (the model predicted "Not at Risk" for a patient who was "At Risk").

    3. Actionable Insight: This analysis revealed our model's primary weakness: it struggles to identify at-risk individuals who are younger and have a lower comorbidity score than the "typical" at-risk patient . This is a clear, actionable insight for future data collection.

- Bonus Analysis (Cell 21):
    - What We Did: We tested our "best model" in a "lightweight" version, using only the top 15 features from our preliminary analysis.
    - Insight: This lightweight model achieved an AUC extremely close to the full 52-feature model. This proves that for a production app, we could use the simpler, faster, and more explainable 15-feature model with minimal performance loss.

Tools Used

- Core Libraries: Python, Pandas, NumPy
- Preprocessing & Modeling: Scikit-learn (for GridSearchCV, Pipeline, StandardScaler, OneHotEncoder, SimpleImputer, StackingClassifier, LogisticRegression, RandomForest, etc.)
- High-Performance Modeling: XGBoost, LightGBM, CatBoost
- Deep Learning: Tensorflow, Keras (Cell 2)
- Visualization: Matplotlib, Seaborn
- Serialization: Joblib

7. GitHub Repo Link

https://github.com/Nivakaran-S/ModelX