Pseudo Code: Parkinson's Disease Motor Progression Prediction Model

Stacking Regressor with Explainable AI (SHAP)

Version: 1.0

Date: November 2025

Performance: R^2=0.551, MAE=6.01, RMSE=7.21 (Independent Clinical Test Set)

Table of Contents

Overview

System Architecture

Plain Text

```
┌─────────────────────────────────────────────────────────────────────┐
│                          TRAINING PIPELINE                          │
├─────────────────────────────────────────────────────────────────────┤
│                                                                     │
│  [1] Load Data → [2] Preprocess → [3] Feature Engineering →       │
│  [4] Split Data → [5] Optimize Hyperparameters →                  │
│  [6] Train Final Model → [7] Validate → [8] SHAP Analysis         │
│                                                                     │
└─────────────────────────────────────────────────────────────────────┘
```

```
┌─────────────────────────────────────────────────────────────────────┐
│                    PREDICTION PIPELINE                              │
├─────────────────────────────────────────────────────────────────────┤
│                                                                     │
│  [1] Load Patient Data → [2] Preprocess → [3] Scale Features →     │
│  [4] Predict → [5] Inverse Transform → [6] Generate Report         │
│                                                                     │
└─────────────────────────────────────────────────────────────────────┘
```

Model Components

- **Base Models:** XGBoost, LightGBM, CatBoost
- **Meta-Learner:** Huber Regressor
- **Explainability:** SHAP (SHapley Additive exPlanations)
- **Validation:** 7-Fold Cross-Validation + Independent Holdout

---

Main Training Pipeline

Algorithm 1: Complete Training Pipeline

Plain Text

```
ALGORITHM: TrainParkinsonsProgressionModel()

INPUT:
    - clinical_data: DataFrame with baseline clinical features
    - rnaseq_data: DataFrame with gene expression data (RNA-seq)
    - random_seed: Integer for reproducibility (default: 42)

OUTPUT:
    - trained_model: Optimized Stacking Regressor
    - performance_metrics: Dictionary with R², MAE, RMSE
    - shap_values: SHAP importance values for features
    - model_artifacts: Scalers, transformers, feature names

PROCEDURE:
    1. INITIALIZE
        SET random_seed = 42
        SET n_top_genes = 100
        SET n_optuna_trials = 30
        SET n_cv_folds = 7
        SET test_size = 0.2
    2. LOAD AND MERGE DATA
        clinical_df ← LOAD_CSV("final_dataset.csv")
        rnaseq_df ← LOAD_CSV("rnaseq_baseline_filtered.csv")
        merged_df ← MERGE(clinical_df, rnaseq_df, on="PATNO")

        PRINT "Loaded", LENGTH(merged_df), "patients"

    3. REMOVE OUTLIERS
        Q1 ← QUANTILE(merged_df["UPDRS_V04"], 0.25)
        Q3 ← QUANTILE(merged_df["UPDRS_V04"], 0.75)
        IQR ← Q3 - Q1

        outlier_mask ← (UPDRS_V04 < Q1 - 1.5×IQR) OR (UPDRS_V04 > Q3 +
1.5×IQR)
        clean_df ← merged_df[NOT outlier_mask]

        PRINT "Removed", SUM(outlier_mask), "outliers"

    4. SELECT TOP GENES
        gene_correlations ← []

        FOR EACH gene IN gene_columns:
            corr ← CORRELATION(clean_df[gene], clean_df["DELTA_UPDRS"])
            IF NOT IS_NAN(corr):
                APPEND (gene, ABS(corr)) TO gene_correlations

        SORT gene_correlations BY correlation DESC
        top_genes ← FIRST 100 genes FROM gene_correlations

        PRINT "Selected top 100 genes with correlation range:",
              MAX(corr), "to", MIN(corr)

    5. ENGINEER FEATURES
        // Clinical features
        clinical_features ← ["UPDRS_BL", "AGE", "GENDER"]

        // PD risk genes
        pd_genes ← ["PD_SNCA", "PD_LRRK2", "PD_GBA", "PD_PRKN",
                    "PD_PINK1", "PD_PARK7", "PD_VPS35"]

        // Pathway scores
        pathway_features ← ["PATHWAY_Inflammation",
                           "PATHWAY_Mitochondrial",
                           "PATHWAY_Autophagy"]

        // Create interaction features
        clean_df["PINK1_x_PARK7"] ← clean_df["PD_PINK1"] ×
```

```
clean_df["PD_PARK7"]
        clean_df["AGE_x_PINK1"] ← clean_df["AGE"] × clean_df["PD_PINK1"]
        clean_df["UPDRS_BL_x_PINK1"] ← clean_df["UPDRS_BL"] ×
clean_df["PD_PINK1"]

        interaction_features ← ["PINK1_x_PARK7", "AGE_x_PINK1",
"UPDRS_BL_x_PINK1"]

        // Combine all features
        all_features ← clinical_features + top_genes + pd_genes +
                       pathway_features + interaction_features

        PRINT "Total features:", LENGTH(all_features)

    6. PREPARE DATA
        X ← clean_df[all_features]
        y_v04 ← clean_df["UPDRS_V04"]  // Target: 12-month UPDRS
        y_clf ← (clean_df["DELTA_UPDRS"] >= 5)  // For stratification

        // Transform target variable
        target_transformer ← PowerTransformer(method="yeo-johnson")
        y_transformed ← target_transformer.FIT_TRANSFORM(y_v04)

    7. SPLIT DATA (Stratified)
        (X_trainval, X_test,
         y_trainval_trans, y_test_trans,
         y_trainval_orig, y_test_orig) ← TRAIN_TEST_SPLIT(
            X, y_transformed, y_v04,
            test_size=0.2,
            stratify=y_clf,
            random_state=random_seed
        )

        PRINT "Train+Val:", LENGTH(X_trainval), "Test:", LENGTH(X_test)

    8. OPTIMIZE HYPERPARAMETERS
        best_params ← OPTIMIZE_HYPERPARAMETERS(
            X_trainval, y_trainval_trans,
            n_trials=30,
            n_folds=7
        )

        PRINT "Best CV R²:", best_params["cv_r2"]

    9. TRAIN FINAL MODEL
        // Scale features
        scaler ← StandardScaler()
        X_trainval_scaled ← scaler.FIT_TRANSFORM(X_trainval)
        X_test_scaled ← scaler.TRANSFORM(X_test)

        // Build ensemble with best hyperparameters
        final_model ← BUILD_STACKING_ENSEMBLE(best_params)

        // Train on full training set
        final_model.FIT(X_trainval_scaled, y_trainval_trans)

        PRINT "Final model trained"

    10. VALIDATE ON TEST SET
        // Predict on test set
        y_test_pred_trans ← final_model.PREDICT(X_test_scaled)

        // Inverse transform to original scale
        y_test_pred ← target_transformer.INVERSE_TRANSFORM(y_test_pred_trans)

        // Calculate metrics
        test_r2 ← R2_SCORE(y_test_orig, y_test_pred)
        test_mae ← MEAN_ABSOLUTE_ERROR(y_test_orig, y_test_pred)
        test_rmse ← SQRT(MEAN_SQUARED_ERROR(y_test_orig, y_test_pred))

        PRINT "Test R²:", test_r2
        PRINT "Test MAE:", test_mae
        PRINT "Test RMSE:", test_rmse

    11. COMPUTE SHAP VALUES
        shap_values ← COMPUTE_SHAP_VALUES(
            final_model,
            X_test_scaled,
            feature_names=all_features
        )

        PRINT "SHAP analysis complete"

    12. SAVE MODEL ARTIFACTS
        model_package ← {
            "ensemble_model": final_model,
            "scaler": scaler,
            "target_transformer": target_transformer,
            "feature_names": all_features,
            "n_features": LENGTH(all_features),
            "cv_results": best_params,
            "test_results": {
                "r2": test_r2,
                "mae": test_mae,
                "rmse": test_rmse
            },
            "shap_values": shap_values
        }

        SAVE_PICKLE(model_package, "lightweight_optimized_model.pkl")

        PRINT "Model saved successfully"

    13. RETURN
        RETURN model_package

END ALGORITHM
```

Data Preprocessing

Algorithm 2: Data Loading and Cleaning

Plain Text

```
ALGORITHM: LoadAndCleanData()

INPUT:
    - clinical_file_path: Path to clinical data CSV
    - rnaseq_file_path: Path to RNA-seq data CSV

OUTPUT:
    - clean_data: Preprocessed DataFrame
    - n_outliers_removed: Number of outliers removed

PROCEDURE:
    1. LOAD DATA
        clinical_df ← READ_CSV(clinical_file_path)
        rnaseq_df ← READ_CSV(rnaseq_file_path)

        PRINT "Clinical data:", SHAPE(clinical_df)
        PRINT "RNA-seq data:", SHAPE(rnaseq_df)

    2. CONVERT PATIENT IDs TO STRING
        clinical_df["PATNO"] ← TO_STRING(clinical_df["PATNO"])
        rnaseq_df["PATNO"] ← TO_STRING(rnaseq_df["PATNO"])

    3. EXTRACT GENE COLUMNS
        gene_columns ← [col FOR col IN rnaseq_df.columns
                       IF col STARTS_WITH "ENSG"]

        rnaseq_genes ← rnaseq_df[["PATNO"] + gene_columns]
        PRINT "Gene columns:", LENGTH(gene_columns)

    4. MERGE DATASETS
        merged_df ← INNER_JOIN(clinical_df, rnaseq_genes, on="PATNO")

        PRINT "Merged patients:", LENGTH(merged_df)

    5. HANDLE MISSING VALUES IN TARGET
        IF ANY(IS_NULL(merged_df["UPDRS_V04"])):
            median_updrs ← MEDIAN(merged_df["UPDRS_V04"])
            merged_df["UPDRS_V04"].FILL_NA(median_updrs)

            PRINT "Filled", SUM(IS_NULL), "missing UPDRS values"

    6. DETECT AND REMOVE OUTLIERS (IQR Method)
        Q1 ← QUANTILE(merged_df["UPDRS_V04"], 0.25)
        Q3 ← QUANTILE(merged_df["UPDRS_V04"], 0.75)
        IQR ← Q3 - Q1

        lower_bound ← Q1 - 1.5 × IQR
        upper_bound ← Q3 + 1.5 × IQR

        outlier_mask ← (merged_df["UPDRS_V04"] < lower_bound) OR
                       (merged_df["UPDRS_V04"] > upper_bound)

        n_outliers ← SUM(outlier_mask)
        clean_df ← merged_df[NOT outlier_mask]

        PRINT "Outliers removed:", n_outliers
        PRINT "Clean data size:", LENGTH(clean_df)

    7. RETURN
        RETURN clean_df, n_outliers

END ALGORITHM
```

Algorithm 3: Gene Selection by Correlation

Plain Text

```
ALGORITHM: SelectTopGenes(data, n_top=100)

INPUT:
    - data: DataFrame with gene expression and DELTA_UPDRS
    - n_top: Number of top genes to select (default: 100)
```

```
OUTPUT:
    - top_genes: List of top gene names
    - correlations: Dictionary mapping genes to correlation values

PROCEDURE:
    1. IDENTIFY GENE COLUMNS
        gene_columns ← [col FOR col IN data.columns
                       IF col STARTS_WITH "ENSG"]

        PRINT "Total genes available:", LENGTH(gene_columns)

    2. COMPUTE CORRELATIONS
        gene_correlations ← []

        FOR EACH gene IN gene_columns:
            IF gene IN data.columns:
                // Compute Pearson correlation with progression
                corr_matrix ← CORRELATION(data[[gene, "DELTA_UPDRS"]])
                corr_value ← corr_matrix[0, 1]

                IF NOT IS_NAN(corr_value):
                    abs_corr ← ABSOLUTE_VALUE(corr_value)
                    APPEND (gene, abs_corr, corr_value) TO gene_correlations

        PRINT "Valid correlations computed:", LENGTH(gene_correlations)

    3. SORT BY ABSOLUTE CORRELATION
        SORT gene_correlations BY abs_corr DESCENDING

    4. SELECT TOP N GENES
        top_genes ← []
        correlations ← {}

        FOR i FROM 0 TO n_top-1:
            gene_name ← gene_correlations[i][0]
            corr_value ← gene_correlations[i][2]

            APPEND gene_name TO top_genes
            correlations[gene_name] ← corr_value

        PRINT "Top gene correlation range:",
              gene_correlations[0][1], "to", gene_correlations[n_top-1][1]

    5. RETURN
        RETURN top_genes, correlations

END ALGORITHM
```

Feature Engineering

Algorithm 4: Feature Engineering Pipeline

Plain Text

```
ALGORITHM: EngineerFeatures(data, top_genes)

INPUT:
    - data: Clean DataFrame
    - top_genes: List of top 100 selected genes

OUTPUT:
    - feature_matrix: Matrix of engineered features
    - feature_names: List of feature names
    - feature_categories: Dictionary categorizing features

PROCEDURE:
    1. DEFINE CLINICAL FEATURES
        clinical_features ← ["UPDRS_BL", "AGE", "GENDER"]

    2. DEFINE PD RISK GENES
        pd_genes ← [
            "PD_SNCA",      // α-synuclein
            "PD_LRRK2",     // Leucine-rich repeat kinase 2
            "PD_GBA",       // Glucocerebrosidase
            "PD_PRKN",      // Parkin
            "PD_PINK1",     // PTEN-induced kinase 1
            "PD_PARK7",     // DJ-1
            "PD_VPS35"      // Vacuolar protein sorting 35
        ]

    3. DEFINE PATHWAY SCORES
        pathway_features ← [
            "PATHWAY_Inflammation",     // Neuroinflammation
            "PATHWAY_Mitochondrial",    // Mitochondrial dysfunction
            "PATHWAY_Autophagy"         // Autophagy/mitophagy
        ]

    4. CREATE INTERACTION FEATURES
        // Gene-gene interaction (mitophagy pathway)
        data["PINK1_x_PARK7"] ← data["PD_PINK1"] × data["PD_PARK7"]

        // Age-gene interaction
        data["AGE_x_PINK1"] ← data["AGE"] × data["PD_PINK1"]
        // Clinical-gene interaction (most important feature!)
        data["UPDRS_BL_x_PINK1"] ← data["UPDRS_BL"] × data["PD_PINK1"]

        interaction_features ← [
            "PINK1_x_PARK7",
            "AGE_x_PINK1",
            "UPDRS_BL_x_PINK1"
        ]

    5. COMBINE ALL FEATURES
        all_features ← clinical_features +
                       top_genes +
                       pd_genes +
                       pathway_features +
                       interaction_features

        // Filter to existing columns
        final_features ← [f FOR f IN all_features IF f IN data.columns]

    6. HANDLE MISSING VALUES
        FOR EACH feature IN final_features:
            IF ANY(IS_NULL(data[feature])):
                median_value ← MEDIAN(data[feature])
                data[feature].FILL_NA(median_value)

                PRINT "Filled missing values in", feature

    7. CREATE FEATURE MATRIX
        X ← data[final_features].TO_NUMPY()

    8. CATEGORIZE FEATURES
        feature_categories ← {
            "clinical": [f FOR f IN clinical_features IF f IN
final_features],
            "top_genes": [f FOR f IN top_genes IF f IN final_features],
            "pd_genes": [f FOR f IN pd_genes IF f IN final_features],
            "pathways": [f FOR f IN pathway_features IF f IN final_features],
            "interactions": [f FOR f IN interaction_features IF f IN
final_features]
        }

        PRINT "Feature breakdown:"
        FOR category, features IN feature_categories:
            PRINT "  ", category, ":", LENGTH(features)

        PRINT "Total features:", LENGTH(final_features)

    9. RETURN
```

```
        RETURN X, final_features, feature_categories

END ALGORITHM
```

Model Training

Algorithm 5: Build Stacking Ensemble

Plain Text

```
ALGORITHM: BuildStackingEnsemble(hyperparameters)

INPUT:
    - hyperparameters: Dictionary with optimized hyperparameters

OUTPUT:
    - ensemble_model: Configured Stacking Regressor

PROCEDURE:
    1. CONFIGURE XGBoost
        xgb_model ← XGBRegressor(
            objective = "reg:pseudohubererror",  // Robust to outliers
            n_estimators = hyperparameters["xgb_n_estimators"],
            max_depth = hyperparameters["xgb_max_depth"],
            learning_rate = hyperparameters["xgb_learning_rate"],
            subsample = hyperparameters["xgb_subsample"],
            colsample_bytree = hyperparameters["xgb_colsample_bytree"],
            min_child_weight = hyperparameters["xgb_min_child_weight"],
            reg_alpha = hyperparameters["xgb_reg_alpha"],  // L1
regularization
            reg_lambda = hyperparameters["xgb_reg_lambda"],  // L2
regularization
            random_state = 42,
            n_jobs = 2
        )

    2. CONFIGURE LightGBM
        lgbm_model ← LGBMRegressor(
            objective = "huber",  // Robust loss function
            n_estimators = hyperparameters["lgbm_n_estimators"],
            max_depth = hyperparameters["lgbm_max_depth"],
            learning_rate = hyperparameters["lgbm_learning_rate"],
            subsample = hyperparameters["lgbm_subsample"],
            colsample_bytree = hyperparameters["lgbm_colsample_bytree"],
            min_data_in_leaf = hyperparameters["lgbm_min_data_in_leaf"],
            reg_alpha = hyperparameters["lgbm_reg_alpha"],
            reg_lambda = hyperparameters["lgbm_reg_lambda"],
            random_state = 42,
            n_jobs = 2
        )

    3. CONFIGURE CatBoost
        catboost_model ← CatBoostRegressor(
            loss_function = "RMSE",
            iterations = hyperparameters["cat_iterations"],
            depth = hyperparameters["cat_depth"],
            learning_rate = hyperparameters["cat_learning_rate"],
            subsample = hyperparameters["cat_subsample"],
            reg_lambda = hyperparameters["cat_reg_lambda"],
            random_state = 42,
            thread_count = 2,
            verbose = False
        )

    4. DEFINE BASE MODELS
        base_models ← [
            ("xgb", xgb_model),
            ("lgbm", lgbm_model),
            ("catboost", catboost_model)
        ]

    5. CONFIGURE META-LEARNER (Huber Regressor)
        meta_learner ← HuberRegressor(
            epsilon = hyperparameters["meta_epsilon"],  // Robustness
parameter
            alpha = hyperparameters["meta_alpha"],      // Regularization
            max_iter = 300
        )

    6. BUILD STACKING ENSEMBLE
        ensemble_model ← StackingRegressor(
            estimators = base_models,
            final_estimator = meta_learner,
            cv = 5,  // Internal cross-validation for meta-features
            n_jobs = 2
        )

        PRINT "Stacking ensemble configured:"
        PRINT "  Base models: XGBoost, LightGBM, CatBoost"
        PRINT "  Meta-learner: Huber Regressor"

    7. RETURN
        RETURN ensemble_model
```

```
END ALGORITHM
```

Hyperparameter Optimization

Algorithm 6: Bayesian Hyperparameter Optimization (Optuna)

Plain Text

```
ALGORITHM: OptimizeHyperparameters(X_train, y_train, n_trials=30, n_folds=7)

INPUT:
    - X_train: Training feature matrix
    - y_train: Training target (transformed)
    - n_trials: Number of Optuna trials (default: 30)
    - n_folds: Number of CV folds (default: 7)

OUTPUT:
    - best_params: Dictionary with best hyperparameters
    - best_cv_score: Best cross-validation R² score

PROCEDURE:
    1. DEFINE OBJECTIVE FUNCTION
        FUNCTION objective(trial):
            // Sample hyperparameters for XGBoost
            xgb_params ← {
                "n_estimators": trial.SUGGEST_INT("xgb_n_estimators", 100,
250),
                "max_depth": trial.SUGGEST_INT("xgb_max_depth", 3, 7),
                "learning_rate": trial.SUGGEST_FLOAT("xgb_learning_rate",
0.01, 0.1, log=True),
                "subsample": trial.SUGGEST_FLOAT("xgb_subsample", 0.6, 0.9),
                "colsample_bytree":
trial.SUGGEST_FLOAT("xgb_colsample_bytree", 0.6, 0.9),
                "min_child_weight":
trial.SUGGEST_INT("xgb_min_child_weight", 1, 8),
                "reg_alpha": trial.SUGGEST_FLOAT("xgb_reg_alpha", 0.01, 1.0,
log=True),
                "reg_lambda": trial.SUGGEST_FLOAT("xgb_reg_lambda", 0.1,
5.0, log=True)
            }

            // Sample hyperparameters for LightGBM
            lgbm_params ← {
                "n_estimators": trial.SUGGEST_INT("lgbm_n_estimators", 100,
250),
                "max_depth": trial.SUGGEST_INT("lgbm_max_depth", 3, 7),
                "learning_rate": trial.SUGGEST_FLOAT("lgbm_learning_rate",
0.01, 0.1, log=True),
                "subsample": trial.SUGGEST_FLOAT("lgbm_subsample", 0.6, 0.9),
                "colsample_bytree":
trial.SUGGEST_FLOAT("lgbm_colsample_bytree", 0.6, 0.9),
                "min_data_in_leaf":
trial.SUGGEST_INT("lgbm_min_data_in_leaf", 5, 25),
                "reg_alpha": trial.SUGGEST_FLOAT("lgbm_reg_alpha", 0.01,
1.0, log=True),
                "reg_lambda": trial.SUGGEST_FLOAT("lgbm_reg_lambda", 0.1,
5.0, log=True)
            }

            // Sample hyperparameters for CatBoost
            catboost_params ← {
                "iterations": trial.SUGGEST_INT("cat_iterations", 100, 250),
                "depth": trial.SUGGEST_INT("cat_depth", 3, 7),
                "learning_rate": trial.SUGGEST_FLOAT("cat_learning_rate",
0.01, 0.1, log=True),
                "subsample": trial.SUGGEST_FLOAT("cat_subsample", 0.6, 0.9),
                "reg_lambda": trial.SUGGEST_FLOAT("cat_reg_lambda", 0.1,
5.0, log=True)
            }

            // Sample meta-learner hyperparameters
            meta_alpha ← trial.SUGGEST_FLOAT("meta_alpha", 0.01, 1.0,
log=True)
            meta_epsilon ← trial.SUGGEST_FLOAT("meta_epsilon", 1.0, 2.0)

            // Build ensemble with sampled hyperparameters
            ensemble ← BUILD_STACKING_ENSEMBLE({
                **xgb_params, **lgbm_params, **catboost_params,
                "meta_alpha": meta_alpha, "meta_epsilon": meta_epsilon
            })

            // Perform k-fold cross-validation
            cv ← KFold(n_splits=n_folds, shuffle=True, random_state=42)
            cv_scores ← []

            FOR train_idx, val_idx IN cv.SPLIT(X_train):
                X_train_fold ← X_train[train_idx]
                y_train_fold ← y_train[train_idx]
                X_val_fold ← X_train[val_idx]
                y_val_fold ← y_train[val_idx]

                // Scale features
                scaler ← StandardScaler()
                X_train_scaled ← scaler.FIT_TRANSFORM(X_train_fold)
                X_val_scaled ← scaler.TRANSFORM(X_val_fold)

                // Train and evaluate
                ensemble.FIT(X_train_scaled, y_train_fold)
                y_val_pred ← ensemble.PREDICT(X_val_scaled)

                // Inverse transform predictions
                y_val_pred_orig ←
target_transformer.INVERSE_TRANSFORM(y_val_pred)
                y_val_orig ← target_transformer.INVERSE_TRANSFORM(y_val_fold)

                // Calculate R²
                r2 ← R2_SCORE(y_val_orig, y_val_pred_orig)
                APPEND r2 TO cv_scores

            // Return mean CV score
            RETURN MEAN(cv_scores)

        END FUNCTION

    2. CREATE OPTUNA STUDY
        study ← optuna.CREATE_STUDY(
            direction = "maximize",  // Maximize R²
            study_name = "lightweight_optimization"
        )

    3. RUN OPTIMIZATION
        PRINT "Starting Bayesian optimization with", n_trials, "trials..."

        study.OPTIMIZE(
            objective,
            n_trials = n_trials,
            show_progress_bar = True
        )

        PRINT "Optimization complete!"

    4. EXTRACT BEST PARAMETERS
        best_params ← study.best_params
        best_cv_score ← study.best_value

        PRINT "Best CV R²:", best_cv_score
        PRINT "Best hyperparameters:"
        FOR param, value IN best_params:
            PRINT "  ", param, ":", value
    5. RETURN
        RETURN best_params, best_cv_score

END ALGORITHM
```

Model Validation

Algorithm 7: Model Validation on Independent Test Set

Plain Text

```
ALGORITHM: ValidateModel(model, X_test, y_test, scaler, target_transformer)

INPUT:
    - model: Trained ensemble model
    - X_test: Test feature matrix (unscaled)
    - y_test: Test target (original scale)
    - scaler: Fitted StandardScaler
    - target_transformer: Fitted PowerTransformer

OUTPUT:
    - metrics: Dictionary with R², MAE, RMSE, Pearson r
    - predictions: Array of predicted values
    - residuals: Array of residuals (actual - predicted)

PROCEDURE:
    1. SCALE TEST FEATURES
        X_test_scaled ← scaler.TRANSFORM(X_test)

    2. MAKE PREDICTIONS (Transformed Space)
        y_test_pred_trans ← model.PREDICT(X_test_scaled)

    3. INVERSE TRANSFORM TO ORIGINAL SCALE
        y_test_pred ← target_transformer.INVERSE_TRANSFORM(
            y_test_pred_trans.RESHAPE(-1, 1)
        ).FLATTEN()

    4. CALCULATE REGRESSION METRICS
        // R² (coefficient of determination)
        r2 ← R2_SCORE(y_test, y_test_pred)

        // MAE (mean absolute error)
        mae ← MEAN_ABSOLUTE_ERROR(y_test, y_test_pred)

        // RMSE (root mean squared error)
        mse ← MEAN_SQUARED_ERROR(y_test, y_test_pred)
        rmse ← SQRT(mse)

        // Pearson correlation coefficient
        pearson_r ← PEARSON_CORRELATION(y_test, y_test_pred)

        // Spearman rank correlation
        spearman_rho ← SPEARMAN_CORRELATION(y_test, y_test_pred)

    5. CALCULATE RESIDUALS
        residuals ← y_test - y_test_pred

        // Residual statistics
        mean_residual ← MEAN(residuals)
        std_residual ← STD(residuals)

        PRINT "Mean residual:", mean_residual
        PRINT "Std residual:", std_residual

    6. PRINT RESULTS
        PRINT "=" * 80
        PRINT "INDEPENDENT TEST SET VALIDATION RESULTS"
        PRINT "=" * 80
        PRINT "R² Score:", r2
        PRINT "MAE:", mae, "UPDRS points"
        PRINT "RMSE:", rmse, "UPDRS points"
        PRINT "Pearson r:", pearson_r
        PRINT "Spearman ρ:", spearman_rho
        PRINT "=" * 80

    7. PACKAGE METRICS
        metrics ← {
            "r2": r2,
            "mae": mae,
            "rmse": rmse,
            "pearson_r": pearson_r,
            "spearman_rho": spearman_rho,
            "mean_residual": mean_residual,
            "std_residual": std_residual,
            "n_samples": LENGTH(y_test)
        }

    8. RETURN
        RETURN metrics, y_test_pred, residuals

END ALGORITHM
```

SHAP Analysis

Algorithm 8: SHAP Feature Importance Analysis

Plain Text

```
ALGORITHM: ComputeSHAPValues(model, X_test, feature_names)

INPUT:
    - model: Trained ensemble model
    - X_test: Test feature matrix (scaled)
    - feature_names: List of feature names

OUTPUT:
    - shap_values: SHAP importance values for each feature
    - feature_importance: Sorted list of (feature, importance) tuples

PROCEDURE:
    1. CREATE SHAP EXPLAINER
        // Use TreeExplainer for gradient boosting models
        explainer ← shap.TreeExplainer(model)

        PRINT "SHAP explainer created"

    2. COMPUTE SHAP VALUES
        PRINT "Computing SHAP values (this may take a few minutes)..."

        shap_values ← explainer.SHAP_VALUES(X_test)

        PRINT "SHAP values computed for", LENGTH(X_test), "samples"

    3. CALCULATE MEAN ABSOLUTE SHAP VALUES
        mean_abs_shap ← []

        FOR i FROM 0 TO LENGTH(feature_names)-1:
            feature_name ← feature_names[i]
            shap_column ← shap_values[:, i]

            mean_abs_value ← MEAN(ABSOLUTE_VALUE(shap_column))

            APPEND (feature_name, mean_abs_value) TO mean_abs_shap

    4. SORT BY IMPORTANCE
        SORT mean_abs_shap BY mean_abs_value DESCENDING

    5. PRINT TOP 20 FEATURES
        PRINT "=" * 80
        PRINT "TOP 20 FEATURES BY SHAP IMPORTANCE"
        PRINT "=" * 80

        FOR i FROM 0 TO 19:
            feature, importance ← mean_abs_shap[i]
            PRINT i+1, ".", feature, ":", importance

        PRINT "=" * 80

    6. CATEGORIZE FEATURES
        // Separate by feature type
        clinical_shap ← []
        gene_shap ← []
        pd_gene_shap ← []
        pathway_shap ← []
        interaction_shap ← []

        FOR feature, importance IN mean_abs_shap:
            IF feature IN ["UPDRS_BL", "AGE", "GENDER"]:
                APPEND (feature, importance) TO clinical_shap
            ELSE IF feature STARTS_WITH "ENSG":
                APPEND (feature, importance) TO gene_shap
            ELSE IF feature STARTS_WITH "PD_":
                APPEND (feature, importance) TO pd_gene_shap
            ELSE IF feature STARTS_WITH "PATHWAY_":
                APPEND (feature, importance) TO pathway_shap
            ELSE IF feature CONTAINS "_x_":
                APPEND (feature, importance) TO interaction_shap

    7. PRINT CATEGORY SUMMARIES
        PRINT "SHAP IMPORTANCE BY CATEGORY:"
        PRINT "  Clinical features:", SUM([imp FOR _, imp IN clinical_shap])
        PRINT "  Top genes:", SUM([imp FOR _, imp IN gene_shap[:20]])
        PRINT "  PD risk genes:", SUM([imp FOR _, imp IN pd_gene_shap])
        PRINT "  Pathways:", SUM([imp FOR _, imp IN pathway_shap])
        PRINT "  Interactions:", SUM([imp FOR _, imp IN interaction_shap])

    8. RETURN
        RETURN shap_values, mean_abs_shap

END ALGORITHM
```

Clinical Prediction

Algorithm 9: Predict for New Patient

Plain Text

```
ALGORITHM: PredictNewPatient(patient_data, model_package)

INPUT:
    - patient_data: Dictionary with baseline clinical data
      {
          "PATNO": "PATIENT_001",
          "UPDRS_BL": 20.0,
          "AGE": 68.0,
          "GENDER": 1.0  // 0=Female, 1=Male
      }
    - model_package: Loaded model artifacts

OUTPUT:
    - prediction_report: Dictionary with predictions and interpretation

PROCEDURE:
    1. EXTRACT MODEL ARTIFACTS
        model ← model_package["ensemble_model"]
        scaler ← model_package["scaler"]
        target_transformer ← model_package["target_transformer"]
        feature_names ← model_package["feature_names"]
        n_features ← model_package["n_features"]
        test_mae ← model_package["test_results"]["mae"]

    2. CREATE PATIENT DATAFRAME
        patient_df ← DataFrame([patient_data])

        PRINT "Patient ID:", patient_data["PATNO"]
        PRINT "Baseline UPDRS:", patient_data["UPDRS_BL"]
        PRINT "Age:", patient_data["AGE"]
        PRINT "Gender:", "Male" IF patient_data["GENDER"]==1 ELSE "Female"

    3. IMPUTE MISSING FEATURES
        // For clinical prediction, we only have baseline clinical data
        // All gene expression and pathway features are imputed with 0

        FOR feature IN feature_names:
            IF feature NOT IN patient_df.columns:
                patient_df[feature] ← 0.0

        // Ensure correct column order
        patient_df ← patient_df[feature_names]

        PRINT "Warning: Missing", n_features - 4, "features (imputed with
zeros)"
```

```
    4. SCALE FEATURES
        X_patient ← patient_df.TO_NUMPY()
        X_patient_scaled ← scaler.TRANSFORM(X_patient)

    5. MAKE PREDICTION (Transformed Space)
        y_pred_trans ← model.PREDICT(X_patient_scaled)

    6. INVERSE TRANSFORM TO UPDRS SCALE
        y_pred_updrs ← target_transformer.INVERSE_TRANSFORM(
            y_pred_trans.RESHAPE(-1, 1)
        ).FLATTEN()[0]

        PRINT "Predicted UPDRS at 12 months:", y_pred_updrs

    7. CALCULATE PREDICTED CHANGE
        baseline_updrs ← patient_data["UPDRS_BL"]
        predicted_change ← y_pred_updrs - baseline_updrs

        PRINT "Predicted change:", predicted_change, "points"

    8. CALCULATE CONFIDENCE INTERVAL
        // Use MAE as uncertainty estimate
        lower_bound ← y_pred_updrs - test_mae
        upper_bound ← y_pred_updrs + test_mae

        PRINT "95% Confidence Interval: [", lower_bound, ",", upper_bound,
"]"

    9. CATEGORIZE PROGRESSION RISK
        IF predicted_change < 0:
            risk_category ← "Improvement"
            confidence ← "Low"  // Unusual, low confidence
        ELSE IF predicted_change < 3:
            risk_category ← "Stable"
            confidence ← "High"
        ELSE IF predicted_change < 5:
            risk_category ← "Mild Progression"
            confidence ← "High"
        ELSE IF predicted_change < 10:
            risk_category ← "Moderate Progression"
            confidence ← "High"
        ELSE:
            risk_category ← "Rapid Progression"
            confidence ← "High"

        PRINT "Progression Risk:", risk_category
        PRINT "Confidence:", confidence
    10. GENERATE CLINICAL INTERPRETATION
        IF risk_category == "Stable":
            interpretation ← "Patient likely to remain stable over 12 months
(" +
                            predicted_change + " points change)."

        ELSE IF risk_category == "Mild Progression":
            interpretation ← "Mild progression expected (" +
predicted_change +
                            " points). Standard monitoring recommended."

        ELSE IF risk_category == "Moderate Progression":
            interpretation ← "Moderate progression expected (" +
predicted_change +
                            " points). Consider treatment adjustment."

        ELSE IF risk_category == "Rapid Progression":
            interpretation ← "Rapid progression expected (" +
predicted_change +
                            " points). Urgent clinical review recommended."

        ELSE:  // Improvement
            interpretation ← "Patient shows predicted improvement (" +
                            predicted_change + " points). Monitor for
accuracy."

        PRINT "Clinical Interpretation:", interpretation

    11. PACKAGE PREDICTION REPORT
        prediction_report ← {
            "patient_id": patient_data["PATNO"],
            "baseline_updrs": baseline_updrs,
            "predicted_updrs_12m": y_pred_updrs,
            "predicted_change": predicted_change,
            "lower_bound_12m": lower_bound,
            "upper_bound_12m": upper_bound,
            "progression_risk": risk_category,
            "confidence_level": confidence,
            "clinical_interpretation": interpretation
        }

    12. RETURN
        RETURN prediction_report

END ALGORITHM
```

Summary Statistics

Key Performance Metrics

Plain Text

```
┌──────────────────────────────────────────────────────────────────┐
│                   MODEL PERFORMANCE SUMMARY                      │
├──────────────────────────────────────────────────────────────────┤
│                                                                  │
│  Training Set (n=312, 7-Fold CV):                               │
│    R² = 0.513 ± 0.052                                           │
│    MAE = 6.15 ± 0.25 UPDRS points                               │
│    RMSE = 7.86 ± 0.57 UPDRS points                              │
│                                                                  │
│  Independent Test Set (n=78):                                    │
│    R² = 0.551 (55.1% variance explained)                        │
│    MAE = 6.01 UPDRS points                                      │
│    RMSE = 7.21 UPDRS points                                     │
│    Pearson r = 0.74                                             │
│                                                                  │
│  Top 3 Features (SHAP):                                          │
│    1. UPDRS_BL × PINK1 = 0.283                                  │
│    2. UPDRS_BL = 0.258                                          │
│    3. ENSG00000243053 = 0.025                                   │
│                                                                  │
└──────────────────────────────────────────────────────────────────┘
```

Computational Complexity

Plain Text

```
┌──────────────────────────────────────────────────────────────────┐
│                   COMPUTATIONAL COMPLEXITY                       │
├──────────────────────────────────────────────────────────────────┤
│                                                                  │
│  Training Phase:                                                 │
│    Time Complexity: O(n × m × t × k)                            │
│      n = number of samples (312)                                │
│      m = number of features (116)                               │
│      t = number of trees per model (~200)                       │
│      k = number of CV folds (7)                                 │
│                                                                  │
│    Space Complexity: O(n × m + t × m)                           │
│                                                                  │
│    Actual Training Time: ~1 hour (30 Optuna trials)             │
│                                                                  │
│  Prediction Phase:                                               │
│    Time Complexity: O(m × t)                                     │
│    Space Complexity: O(m)                                        │
│                                                                  │
│    Actual Prediction Time: <1 second per patient                 │
│                                                                  │
│  Model Size: 582 KB (compressed)                                 │
│                                                                  │
└──────────────────────────────────────────────────────────────────┘
```

End of Pseudo Code Documentation

Version: 1.0

Last Updated: November 15, 2025

Author: Clinical ML Research Team

For implementation details, see the actual Python code in `lightweight_optimization.py`