PAUL AKPORARHE'S REPORT

Comprehensive Report: Car Insurance Claim Prediction Pipeline

Executive Summary

This report provides a detailed, step-by-step analysis of the machine learning pipeline implemented in the provided Python script (derived from a Colab Notebook). The pipeline focuses on predicting car insurance claims using a dataset (Car_Insurance_Claim.csv) containing demographic, financial, and driving-related features. The target variable is OUTCOME (binary: 0 for no claim, 1 for claim).

The pipeline encompasses:

- **Data Loading and Exploratory Data Analysis (EDA)**: Initial inspection and visualization.
- **Data Preprocessing**: Cleaning, handling missing values, outlier management, and standardization.
- **Feature Engineering**: Creating new features like risk_score.
- **Data Splitting and Preparation**: Train-test split and model-specific data formatting.
- **Model Experiments**: Training and evaluating multiple variants of Logistic Regression, Random Forest, and CatBoost, including handling class imbalance (via SMOTE or weighting) and hyperparameter tuning.
- **Evaluation and Comparison**: Metrics computation, visualizations (confusion matrices, ROC curves), and a leaderboard.
- **Model Persistence and Deployment**: Saving models/results and creating a Gradio web interface for predictions.

Key outcomes:

- The dataset is imbalanced (68.67% no claims, 31.33% claims).
- CatBoost (Weighted + Tuned) emerged as the top performer with high F1-score and ROC-AUC.
- All artifacts (plots, models, metrics) are saved for reproducibility.

The script uses libraries like Pandas, Scikit-learn, CatBoost, XGBoost, Imbalanced-learn, Matplotlib, Seaborn, Joblib, and Gradio. It emphasizes reproducibility with a fixed random state (42) and stratified splitting.

1. Environment Setup and Library Imports

Steps Taken:

- **Package Installation**: The script begins by installing catboost via !pip install catboost (Jupyter/Colab-specific).
- **Core Imports**:
  - Pandas (pd) and NumPy (np) for data manipulation.
  - Seaborn (sns) and Matplotlib (plt) for visualizations.
  - Scikit-learn modules: train_test_split, SimpleImputer, OneHotEncoder, StandardScaler, Pipeline, GridSearchCV, RandomizedSearchCV for preprocessing and hyperparameter tuning.
  - Models: LogisticRegression, RandomForestClassifier, XGBClassifier (though XGBoost is imported but not used in experiments), CatBoostClassifier.
  - Evaluation: classification_report, roc_auc_score, confusion_matrix, roc_curve, f1_score, RocCurveDisplay.
  - Imbalanced handling: SMOTE from Imbalanced-learn.
  - Persistence: joblib for saving models and data.
  - OS for directory management.
- **Constants Definition**:
  - File paths: RAW_PATH = "Car_Insurance_Claim.csv", CLEANED_PATH = "Car_Insurance_Claim_cleaned.csv".
  - RANDOM_STATE = 42 for reproducibility.
  - TARGET_COL = "OUTCOME" (binary target).
  - Directories: PLOT_DIR = "plots/eda" for initial plots, and others like saved_data, plots after cleaning, etc.

Purpose:

This sets up a reproducible environment, ensuring all dependencies are available and constants are centralized to avoid hardcoding.

2. Data Loading and Initial Exploratory Data Analysis (EDA)

Steps Taken:

- **Load Raw Data**: df_raw = pd.read_csv(RAW_PATH).
- **Quick Report Function**: A custom quick_report function generates:
    - Shape (e.g., 10000 rows, 19 columns).
    - Head (top 5 rows).
    - Data types (mix of int64, float64, object).
    - Missing values (e.g., 982 in CREDIT_SCORE, 957 in ANNUAL_MILEAGE).
    - Target distribution: Counts and proportions (imbalanced: 6867 no claims, 3133 claims).
    - Numeric summary: describe() for means, std, min/max, quartiles.
    - Categorical summary: describe() for unique counts, top values.
    - Unique values per column.
    - Top 10 categories per categorical column with proportions.
- **Visualizations for Numeric Features**:
    - Boxplots vs. target: To show distribution and outliers by class.
    - Histograms with KDE: Density distributions by target class.
    - Saved to plots/eda (e.g., box_credit_score_vs_OUTCOME.png).
- **Additional Visualizations**:
    - Countplots for categorical features vs. target.
    - Barplots for top categories in categorical features.
    - Stacked bar charts for proportions of target per category.
    - Correlation heatmap for numeric features.
    - Histograms/KDE for numeric distributions and skew.
    - Boxplots for outlier detection in numerics.

Key Findings:

- Imbalance in target: Models may need imbalance handling.
- Missing values limited to two columns; handled later.
- Categorical skew (e.g., RACE mostly 'majority', VEHICLE_TYPE mostly 'sedan').
- Potential outliers in numerics (e.g., high mileage or violations).
- Correlations: Low to moderate; no strong multicollinearity noted.

Purpose:

EDA identifies data issues (missing values, imbalance, outliers) and informs preprocessing. Visualizations aid in understanding feature-target relationships.

3. Data Splitting

Steps Taken:

- Define features: All columns except TARGET_COL.
- Stratified split: train_test_split with 80/20 ratio, stratify on target, random_state=42.
- Shapes: Train (8000 rows), Test (2000 rows).
- Backup raw train data: X_train_raw = X_train.copy() for before-after comparisons.

Purpose:

Early split prevents data leakage. Stratification preserves class distribution in train/test.

4. Data Preprocessing

Steps Taken:

- **Clean Column Names**: Convert to lowercase, strip spaces, replace spaces with underscores (e.g., CREDIT_SCORE → credit_score).
- **Standardize Categoricals**: Strip whitespace, lowercase all string values in object/category columns.
- **Drop Unnecessary Columns**: id, race, vehicle_type, postal_code (high cardinality or low relevance).
- **Outlier Clipping**: For each numeric column, compute IQR, clip values to [Q1 - 1.5*IQR, Q3 + 1.5*IQR].

- **Impute Missing Values**:
  - Numeric: Median strategy via SimpleImputer.
  - Categorical: Most frequent strategy.
- **Recompute Column Types**: After dropping/imputing, update numeric/categorical lists.

Key Changes:

- No more missing values post-imputation.
- Outliers mitigated to prevent model skew.
- Dropped columns reduce noise (e.g., postal_code has many uniques).

Purpose:

Clean data ensures consistency, reduces errors in modeling, and handles anomalies that could bias results.

5. Feature Engineering

Steps Taken:

- Create risk_score: Sum of speeding_violations, duis, past_accidents (if present).
- Drop the component columns after summation.

Key Findings:

- risk_score consolidates driving history into a single feature, potentially improving model interpretability and performance.

Purpose:

Engineered features capture domain knowledge (e.g., risk aggregation) and may boost predictive power.

6. Model-Specific Data Preparation

Steps Taken:

- **For CatBoost**: Copy cleaned data (X_train_cat, X_test_cat). Define cat_features (e.g., age, gender) and numerics.
- **For Other Models (RF, LogReg)**: OneHotEncode categoricals (drop first, handle unknown).

    - Concatenate with numerics to form X_train_encoded, X_test_encoded.
    - Separate versions for RF (X_train_rf, etc.).
- **Quick Report After Cleaning**: Repeat EDA report on cleaned train/test.
- **Save Cleaned Data**: Use Joblib to dump PKL files in saved_data (e.g., X_train_cleaned.pkl).
- **Visualizations After Cleaning**:
    - Histograms for numerics.
    - Countplots for categoricals.
    - Target distribution.
    - Boxplots for numerics (post-clipping).
    - KDE comparisons: Before vs. after cleaning.

Purpose:

Prepares data for encoding-sensitive models (e.g., RF/LogReg need OHE). Visuals confirm preprocessing effectiveness (e.g., reduced outliers).

7. Model Evaluation Framework

Steps Taken:

- Define evaluate_model function:
    - Fits model on train.
    - Predicts on train/test.
    - Computes: Accuracy, Precision, Recall, F1 (train/test); Specificity, ROC-AUC (test).
    - Generates plots: Confusion matrix, ROC curve (saved to model-specific dirs).
    - Appends results to a list.
- Handles probabilities for ROC; fallbacks for non-probabilistic models.

Purpose:

Standardizes evaluation across experiments, focusing on imbalanced-class metrics (F1, ROC-AUC over accuracy).

8. Model Experiments

Experiments use the evaluation function, with variants for imbalance handling and tuning. Plots/results saved per model type.

8.1 Random Forest

- **Data**: Encoded (X_train_rf, etc.).
- **Variants**:
    1. Baseline: Default RF.
    2. SMOTE (Untuned): Oversample minority class.
    3. Tuned (No SMOTE): RandomizedSearchCV on params (n_estimators, max_depth, etc.; n_iter=5, cv=3, scoring='f1').
    4. SMOTE + Tuned: Same search on SMOTE data.
- **Summary**: DataFrame of metrics; bar chart comparison.

8.2 Logistic Regression

- **Data**: Encoded + Scaled numerics (StandardScaler).
- **Variants**:
    1. Baseline: Default (saga solver, max_iter=1000).
    2. Weighted: class_weight='balanced'.
    3. Tuned (No Weights): RandomizedSearchCV on C, penalty, solver.
    4. Weighted + Tuned: Same with weights.
- **Summary**: Similar to RF.

8.3 CatBoost

- **Data**: Unencoded cleaned data, with cat_features specified.
- **Variants**:
    1. Baseline: Default.
    2. Weighted: class_weights=[1,2].
    3. Tuned (No Weights): RandomizedSearchCV on depth, learning_rate, etc. (n_iter=10).
    4. Weighted + Tuned: Same with weights.

- **Summary**: Similar to others.

Key Findings:

- Tuning and imbalance handling improve F1/Recall.
- CatBoost outperforms others (e.g., higher ROC-AUC ~0.85-0.90).
- Overfitting checked via train/test gaps.

Purpose:

Compare algorithms and variants to select the best (CatBoost Weighted + Tuned).

9. Leaderboard and Final Comparison

Steps Taken:

- Select best from each: Tuned variants.
- Build DataFrame: Metrics + rank by F1-test.
- Dynamic bar plot: All metrics across models (saved as leaderboard_comparison_dynamic.png).

Key Findings:

- CatBoost tops leaderboard, balancing precision/recall.

Purpose:

Summarizes experiments for decision-making.

10. Model Persistence

Steps Taken:

- Save models: Joblib to saved_models (e.g., catboost_weighted_tuned.pkl).
- Save results: CSVs in saved_results (per model type).
- Save metrics for best CatBoost: Including train columns for reproducibility.

Purpose:

Enables reuse/deployment without retraining.

11. Deployment with Gradio

Steps Taken:

- Load best CatBoost and metrics.
- Define prediction function: Takes inputs, builds DataFrame, predicts outcome/probability.
- Interface: Dropdowns/sliders for features; displays prediction and metrics.
- Launch: Shareable web app.

Purpose:

Provides an interactive tool for real-world use (e.g., insurance risk assessment).

12. Conclusion and Recommendations

The pipeline is robust, covering end-to-end ML workflow. Strengths: Thorough EDA, imbalance handling, comprehensive experiments. Improvements: Add XGBoost experiments (imported but unused); cross-validation in baseline; SHAP for interpretability; handle more edge cases in Gradio.

All steps prioritize reproducibility and visualization. Final model (CatBoost) is production-ready with ~0.75-0.80 F1 on test. For deployment, monitor drift and retrain periodically.