Name: Cody Mathews

uID: u0712564

email: u0712564@umail.utah.edu

Github Repository Link: https://github.com/cmathew72/ML2024F

ML2024F

Problem Definition and Motivation

This dataset was extracted by Barry Becker from the 1994 Census database. A set of reasonably clean records was extracted using the specific conditions. Prediction task is to determine whether a person makes over 50K a year given the census information. There are 14 attributes, including continuous, categorical and integer types. Some attributes may have **missing** values, recorded as **question marks**.

Income prediction based on census data is a critical task that provides insights into demographic and economic patterns. The specific problem that was worked in this project was predicting whether an individual earns over $50K annually using structured data extracted from the 1994 U.S. Census database. This prediction task has significant applications in areas such as policymaking, targeted marketing, and socio-economic research.

This problem is particularly interesting because it involves a mixture of continuous, categorical, and integer features, requiring careful preprocessing and model selection. The motivation for using machine learning techniques stems from their ability to handle high-dimensional, structured data and extract complex, non-linear relationships among variables. These techniques provide robust predictions, making them suitable for this binary classification task.

My Solution

Data Preprocessing

The first step in the project involved thoroughly preprocessing the dataset to ensure it was suitable for training machine learning models. Several key actions were taken in this phase:

1. **Handling Missing Values**: Some features in the dataset, particularly the workclass, occupation, and native.country, contained missing values, represented as question marks (?). To handle this, I replaced the missing values with the most frequent value (mode) in each of these columns. This approach preserves data while maintaining simplicity.

2. **Feature Encoding**: Since many features in the dataset are categorical (e.g., workclass, education, relationship), I applied one-hot encoding to convert these categories into numerical representations. One-hot encoding is essential because machine learning models typically perform better with numerical input.

3. **Feature Engineering**: To capture more complex relationships in the data, I created new features based on existing ones:

- **Binning Continuous Variables**: I transformed the age and hours.per.week features by creating bins or groups. This helps simplify continuous variables by breaking them into categories. For example, age was binned into groups such as 18-25, 26-35, and so on. Similarly, hours.per.week was binned into meaningful ranges.
- **Interaction Features**: I created interaction terms to capture the relationships between categorical variables. Specifically, I combined workclass and education into a new feature called workclass_education. This helps the model recognize patterns between different combinations of work classes and education levels.

4. **Scaling Features**: After encoding the categorical variables and creating interaction terms, I scaled the numerical features using StandardScaler. This ensures that all features have a mean of 0 and a standard deviation of 1, which is particularly important for models that are sensitive to feature magnitudes, such as logistic regression and XGBoost.

Models/Algorithms/Results

Logistic Regression

The first model I experimented with was logistic regression. Logistic regression is a good starting point because it is interpretable and often performs well on binary classification tasks.

1. **Hyperparameter Tuning**: I used grid search to optimize the hyperparameters for logistic regression. The key parameter I tuned was the regularization strength (C), which controls the amount of regularization applied to the model. I also experimented with both l1 (lasso) and l2 (ridge) regularization to prevent overfitting.
    - The best parameters I found were C=0.1, with l1 regularization.
2. **Performance**: After tuning the logistic regression model, it achieved an AUC of **0.90936** on the validation set. This was a strong result and provided a solid baseline for further improvements.

Random Forest

Next, I experimented with the Random Forest model, which is an ensemble method that builds multiple decision trees and averages their predictions. Random Forest is known for handling non-linear relationships well and often performs better with structured data like this.

1. **Hyperparameter Tuning**: I applied grid search to tune the following hyperparameters:
    - n_estimators: The number of trees in the forest.
    - max_depth: The maximum depth of each tree.
    - min_samples_split: The minimum number of samples required to split an internal node.

The best-performing Random Forest model used 500 estimators, a maximum depth of 20, and a minimum samples split of 5.

2. **Performance**: The Random Forest model achieved an AUC of **0.89458**, which was slightly lower than the logistic regression model. I concluded that while Random Forest performed well,

logistic regression with feature engineering captured the relationships in the data more effectively.

XGBoost

I then implemented an XGBoost model, a popular gradient boosting algorithm known for its strong performance in structured data competitions.

1. **Model Training**: I trained the XGBoost model using the default parameters and evaluated its performance on the validation set. XGBoost automatically handles interactions between features and is highly customizable through hyperparameter tuning.
2. **Performance**: The initial run of XGBoost gave an AUC of **0.902**, slightly outperforming Random Forest but still not reaching the performance of logistic regression. I plan to tune the hyperparameters of XGBoost to improve this result further.

Ensemble Methodology: Stacking

To maximize predictive power, I implemented a stacked ensemble model:

1. **Base Models**: XGBoost, Random Forest, and Logistic Regression were each fine-tuned to optimize their individual AUC scores.
2. **Meta-Model**: Logistic Regression was used to blend the predictions of the base models. Using out-of-fold predictions from each base model, I trained the meta-model on validation predictions, enabling it to capture additional patterns.

Observations

The stacked model achieved an AUC of **0.92710** on my validation set, which significantly outperformed the other individual models. However, the Kaggle public leaderboard score was **0.92468**, reflecting that only 50% of the test set was used for this evaluation.

Future Work

If I had much more time, I would continue working on this project in the following ways:

3. **Advanced Feature Engineering**: Explore techniques such as target encoding for categorical features and creating polynomial interactions for numerical variables.
4. **Alternative Ensemble Techniques**: Experiment with methods like weighted blending and hierarchical stacking to further optimize ensemble performance.
5. **Deep Learning**: Incorporate deep learning models, such as neural networks with embedding layers for categorical features, to capture complex relationships.
6. **Hyperparameter Optimization**: Use automated tools like Optuna to systematically explore the hyperparameter space of base models and the meta-model.
7. **Explainability**: Leverage tools like SHAP to better understand model predictions and feature importance.