Project Report
DSAA2011 Machine Learning

Hua XU
HKUST(GZ)
hxu401@connect.hkust-gz.edu.cn

Jianhao RUAN
HKUST(GZ)
jruan189@connect.hkust-gz.edu.cn

Leyi WU
HKUST(GZ)
lwu398@connect.hkust-gz.edu.cn

1 Introduction

Predicting adult income from various attributes is a canonical binary classification problem with profound social implications. In this work, we employ the Census Income dataset [1], which records variables such as age, education, work class, occupation and hours-per-week, to infer whether an individual's annual income exceeds $50 000.

Our pipeline begins with thorough data preprocessing—imputing missing values, encoding categorical features, and conducting exploratory visualization. In particular, we apply t-distributed stochastic neighbor embedding (t-SNE) to project the high-dimensional data into two dimensions. This analysis reveals (i) high dimensionality induced by one-hot encoding; (ii) poor separability of the two income classes in a low-dimensional embedding; and (iii) indications of sub-clusters within each class. Subsequent feature-importance analysis confirms that many attributes contribute minimally, suggesting significant redundancy.

To uncover latent structure, we perform clustering via K-means and agglomerative hierarchical methods. Although K-means clusters optimize internal cohesion metrics, agglomerative clustering aligns more closely with the true labels, indicating that income classes exhibit complex, non-convex patterns that violate K-means' spherical-cluster assumption. Building on these insights, we evaluate a suite of classifiers—logistic regression, single decision trees, Random Forest and XGBoost—and demonstrate that ensemble techniques, particularly XGBoost, achieve the highest accuracy by iteratively correcting misclassifications and emphasizing key features.

A further challenge is the pronounced class imbalance: 37155 instances labeled "<=50 K" versus 11687 instances labeled ">50 K". Such skew can bias models toward the majority class and degrade performance on the minority. To address this, we investigate under-sampling and over-sampling, with empirical resultings showing the effectiveness of the two apporaches.

The remainder of the report details our preprocessing workflow, clustering and classification methodologies, and a comprehensive analysis of how feature redundancy and class imbalance impact model performance, ultimately establishing XGBoost combined with appropriate sampling as the most effective solution for adult income classification.

2 Methods and results

This section details the sequential pipeline of our data analysis, commencing with data preprocessing, followed by exploratory data visualization, clustering analysis to uncover underlying groupings, and culminating in the training and evaluation of predictive classification models.

39th Conference on Neural Information Processing Systems (NeurIPS 2025).

2.1 Data Preprocessing

Census Income dataset [1] is structured for binary classification and comprises 14 usable variables. These variables include six of integer type and eight of categorical type. Specific preprocessing strategies were applied to handle these different data types:

- **Categorical Variables:** Missing values within the categorical features were imputed using the most frequent value observed for each respective feature. Subsequently, these categorical features were transformed into a numerical format using one-hot encoding, resulting in one-hot vectors for each category. This transformation was performed in Python utilizing the `OneHotEncoder` from the scikit-learn library.
- **Numeric Variables:** For numeric features, missing values were replaced with the median value of the corresponding variable. Following imputation, these numeric features were standardized to have a mean of 0 and a variance of 1. This normalization was achieved using the `StandardScaler` from scikit-learn.

Upon completion of these preprocessing steps—imputing missing values and converting all features to a numeric representation—the dataset expanded to a dataframe with 105 features. This increased dimensionality makes the visualization even harder. To address this challenge, we apply t-SNE dimensionality reduction technique on the processed data, which will be discussed in 2.2.

Moreover, as for the dataset itself, we may see that the number of samples not evenly fall into the two classes, 37155 instances labeled "<=50 K" versus 11687 instances labeled “>50 K”. This may cause troubles when training models, which will be discussed in 2.4.

2.2 Data visualization

The t-SNE dimensionality reduction technique, known for its strength in preserving local similarities between data samples, was employed for initial data exploration through visualization. For better preserve the information within the original data, we are using different hyperparameter settings to perform the dimensionality reduction, and the results are illustrated in Table 1, through which we may see that the perplexity of value 50 can strike a balance between local and global properties, bringing us best visualization effects while not consuming too much computational power. The visualization results are in Figure 1

This visual analysis revealed several key characteristics of the dataset. Firstly, it highlighted the potential presence of multiple sub-clusters, even within the same ground truth class label. This observation suggests considerable diversity within classes, potentially indicating the existence of distinct sub-classes. Such granularity might necessitate fitting different predictive models to these identified sub-clusters should overall model performance prove unsatisfactory with a single global model.

Secondly, and significantly, the t-SNE visualization indicated that samples from the two primary income classes ($\leq 50K$ and $> 50K$) are not clearly distinguishable, often co-occurring within the same visual clusters despite their different labels. Specifically, samples with income less than 50K appear broadly distributed across the visualization, whereas samples with income greater than 50K tend to concentrate on one side of the plot. This pattern suggests that the $\leq 50K$ class is more general and heterogeneous, encompassing a wider range of data distributions, compared to the more localized $> 50K$ class.

Table 1: The relationship between performance of t-SNE (measured with KL divergence to the original distribution) and the target dimension and perplexity. The lower the KL divergence value is, the better the performace is.

		Perplexity			
		5	30	50	100
Dimensionality	2	2.459	1.953	1.877	1.796
	3	1.640	1.648	1.630	1.583

The observed overlap between classes in the t-SNE embedding, while potentially influenced to some extent by information loss inherent in any dimensionality reduction process, strongly implies that many features in the original high-dimensional dataset may not be effective at separating the two income classes. These less discriminative features could be causing samples from different classes to appear close to each other in the original feature space. Consequently, there might be a need to identify and prioritize more important, distinguishing features to enhance model performance, which is confirmed in our later experiments in Section 2.5.

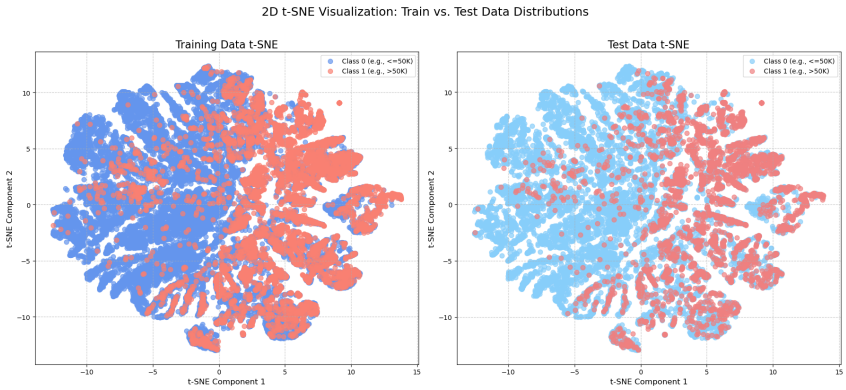


Figure 1: 2D t-SNE Visualization of Test and Train Datasets

2.3 Clustering analysis

To further understand the characteristics of the dataset, we perform clustering analysis; and for simplicity as well as computational efficiency, we are using the 3D t-SNE embedding to perform the analysis.

Methods

Observations from the t-SNE visualization suggested inherent clustering characteristics within the dataset. To further investigate these structures and gain deeper insights, a clustering analysis was performed using two distinct algorithms: K-means and agglomerative hierarchical clustering.

K-means K-means is a widely recognized partitional clustering algorithm that aims to partition N observations into K clusters. The algorithm operates iteratively through two main steps:

Assignment step Each data point xi from the dataset X is assigned to the cluster whose centroid μ_k is closest. This assignment is determined by the formula: $r_i = \underset{k \in \{1 \ldots K\}}{\arg\min} \|x_i - \mu_k\|$.

Update step After all points are assigned to clusters, the centroid of each cluster is recalculated as the mean of all points assigned to it: $\mu_k = \frac{\sum_{i:r_i=k} x_i}{\sum_{i:r_i=k} 1}$. The quality of the initial centroid selection significantly impacts K-means performance. Therefore, the **k-means++** initialization technique was employed. This method selects the first centroid randomly from the data points. Subsequent centroids are chosen from the remaining data points with a probability proportional to their squared distance from the nearest existing centroid, repeating until K centers are selected. This approach helps mitigate sensitivity to poor initializations and theoretically guarantees an $O(\log K)$-competitive solution. The K-means algorithm was implemented in Python using the `KMeans` class from the scikit-learn library.

Agglomerative clustering Agglomerative hierarchical clustering employs /a bottom-up strategy, initially treating each data point as an individual cluster. It then iteratively merges the closest pairs of clusters until only a single cluster containing all data points remains, or a pre-specified number of clusters is reached. The method for measuring the distance between clusters is critical to its performance, and here we are using Ward's method, which aims to minimize the total within-cluster

sum of squares, and the cost of merging two clusters, A and B, can be expressed as: $\Delta(A, B) = \frac{2n_A n_b}{n_A + n_B} \|\mu_A - \mu_B\|^2$ where n_A, n_B represent the number of elements in cluster A and B, respectively. This algorithm was implemented in Python using the `AgglomerativeClustering` in scikit-learn.

The selection of these two methods was motivated by their distinct characteristics. K-means, being a classic and computationally efficient algorithm, provides a rapid baseline for clustering performance. Agglomerative hierarchical clustering was chosen due to observations from the t-SNE analysis (as seen in Figure 1), which suggested the presence of potential sub-clusters. It was hypothesized that a hierarchical approach might better capture the original data distribution and provide more nuanced insights into these finer-grained structures.

Table 2: Comparison of Clustering Algorithm Performance Metrics

Algorithm	Silhouette Score	Calinski-Harabasz Score	Davies-Bouldin Score	Adjusted Rand Score
K-means	0.246131	16804.794922	1.622996	0.080296
Agglomerative Clustering	0.214741	13179.875977	1.678743	0.140342

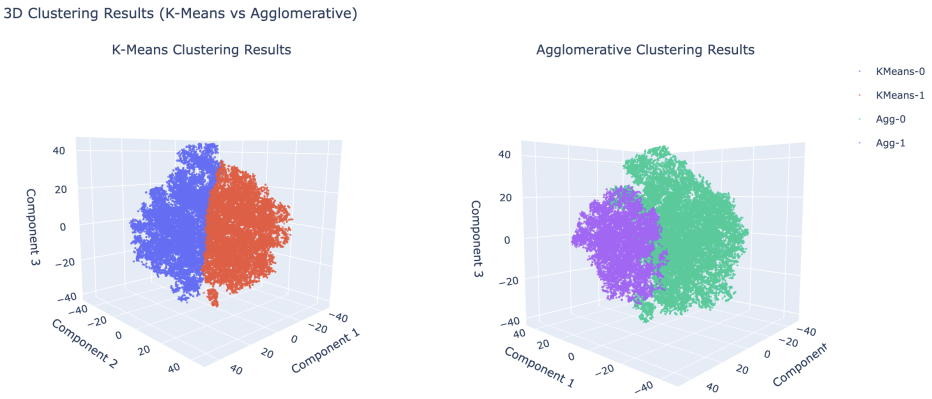


Figure 2: This figure shows the 3D clustering results for K-means (a) and agglomerative hierarchical clustering (b).

Results

The outcomes of the K-means and agglomerative clustering algorithms were visualized in Figure 2 and quantitatively evaluated using several established metrics. To assess the intrinsic quality of the clusters, including their compactness and separation, the Silhouette Score, Calinski-Harabasz Score, and Davies-Bouldin Score were employed. Furthermore, the Adjusted Rand Score (ARI) was utilized as an external metric to evaluate the extent to which the resulting clusters align with the ground truth labels of the dataset.

The analysis of these metrics revealed an interesting divergence in performance. Regarding the internal quality metrics, K-means demonstrated superior performance compared to agglomerative clustering. The higher Silhouette Score and Calinski-Harabasz Score, alongside a lower Davies-Bouldin Score for K-means, indicate that it formed clusters with comparatively better intra-cluster similarity and greater inter-cluster separation from a geometric perspective.

However, when evaluated using the external Adjusted Rand Index, agglomerative hierarchical clustering outperformed K-means. This finding suggests that while K-means may produce geometrically "neater" clusters (often spherical, due to its objective function), these do not align as closely with the actual underlying class distributions as the clusters produced by the hierarchical method. This discrepancy implies that the true data distribution is complex, and an over-reliance on preferred geometric shapes (a characteristic of K-means) might yield high scores on shape-focused internal

metrics but fail to capture the nuanced information present in the ground truth labels. Moreover, the not balanced distribution of the two labels may also be one of the reasons behind the bad performance of K-means, which assumes that the clusters have similar densities and sizes.

The observed diversity within one of the ground truth classes, particularly the tendency for these samples to exhibit sub-clustering (as hinted at in the t-SNE visualization, e.g., the right-hand side of Figure 1), leads to the hypothesis that these sub-structures might be influenced by noisy or less discriminative features that prevent clear separation, making the clustering performance not satisfactory. Agglomerative clustering will be further explored in future analysis to determine if it can more effectively delineate these potential sub-clusters.

2.4 Data Postprocessing

2.5 Prediction: Training and Testing

Methods

Insights derived from the t-SNE visualization and preceding clustering analysis suggest that the dataset exhibits a complex underlying distribution. This complexity implies that a simple linear separator or a hyperplane in the high-dimensional feature space may not be sufficient to accurately distinguish between the classes. Consequently, it is hypothesized that simpler models, such as standard logistic regression, might yield suboptimal performance on this classification task. Therefore, for the prediction phase, decision tree-based models were primarily investigated due to their inherent capability to construct more intricate decision spaces and boundaries. Logistic regression was included as a baseline for comparison against these tree-based methods.

Logistic Regression Logistic Regression is a statistical model used for binary classification tasks, and can be extended for multi-class problems. It models the probability of a binary outcome using a logistic function (sigmoid function) applied to a linear combination of input features. For a given input sample x_t and parameters θ and θ_0 (offset), the probability of observing the class label $y_t \in \{-1, +1\}$ can be expressed as:

$$Pr(y_t|x_t, \theta, \theta_0) = \frac{1}{1 + \exp(-y_t(\langle \theta, x_t \rangle + \theta_0))}$$

The model parameters (θ, θ_0) are typically learned by minimizing a loss function, often the negative log-likelihood, which for n samples can be written as $\sum_{t=1}^{n} \log[1 + \exp(-y_t(\langle x_t, \theta \rangle + \theta_0))]$. To prevent overfitting and improve generalization, regularization techniques such as L_2-norm regularization are commonly incorporated, leading to an objective function like:

$$\min_{(\theta,\theta_0)\in\mathbb{R}^d\times\mathbb{R}} \frac{\lambda}{2}\|\theta\|^2 + \sum_{t=1}^{n} \log[1 + \exp(-y_t(\langle x_t, \theta \rangle + \theta_0))]$$

where λ is the regularization parameter. This optimization is often solved using gradient-based methods. In this project, Logistic Regression is implemented using the `LogisticRegression` class from scikit-learn.

Decision Trees A Decision Tree is a non-parametric supervised learning method used for classification and regression tasks. It functions by constructing a tree-like model of decisions based on the input features. The tree is composed of decision nodes, which represent tests on specific features (e.g., $feature_A > value_X$), branches representing the outcomes of these tests, and leaf nodes (or terminal nodes) which represent the class labels (for classification) or continuous values (for regression). The construction of a decision tree involves a recursive partitioning of the dataset. At each node, the algorithm selects a feature $h^*(x)$ and a split point that best separates the data according to a certain criterion, such as minimizing impurity (e.g., Gini impurity or entropy) or maximizing information gain, which aims to reduce uncertainty at each step. This process continues until a stopping condition is met, for instance, when a node becomes pure (all samples in the node belong to the same class), a predefined maximum depth of the tree is reached, or the number of samples in a node falls below a minimum threshold. The resulting model is a hierarchical structure that is relatively easy to interpret and visualize. Decision tree models are implemented in Python using the `DecisionTreeClassifier` class from scikit-learn.

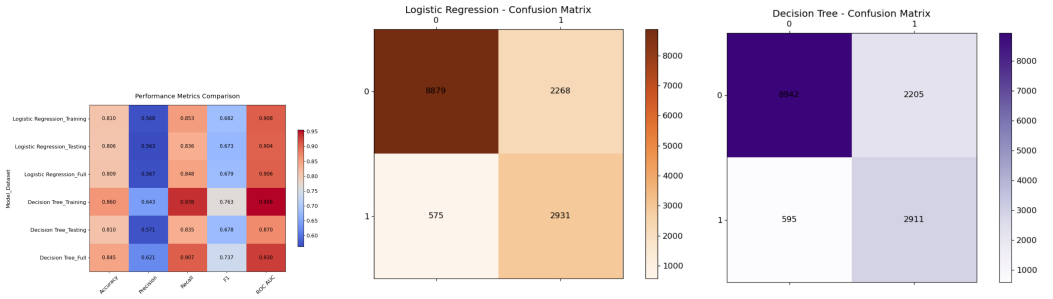
Configuration

In the model training phase, we utilized several distinct classifiers, each carefully configured with specific hyperparameters to optimize their predictive performance and ensure the consistency of our results. For instance, our Random Forest model was constructed using 200 trees, with each tree having a maximum depth of 15 levels and requiring at least 5 samples to form a leaf node; class weights were balanced to address potential dataset imbalances, and a random state of 42 was set for reproducibility, while all available processor cores were engaged for parallel processing. Similarly, the XGBoost classifier was set up with 150 boosting rounds, a maximum tree depth of 6, and a learning rate of 0.1. It utilized 80% of the training data for growing each tree and 80% of features for constructing each tree, incorporated a scale position weight to balance class weights, used AUC as the evaluation metric, started with an initial prediction score of 0.5, maintained a random state of 42, and also leveraged all CPU cores. The Logistic Regression model was configured with balanced class weights to handle imbalanced data, allowed for a maximum of 1000 iterations for convergence, used a random state of 42, and was also set to use all available cores, although its solver might not always gain significant speed-up from this. Finally, our Decision Tree classifier was set with a maximum depth of 15, consistent with the Random Forest's trees, employed balanced class weights, and used a random state of 42 to ensure reproducible outcomes. These parameter choices were made to strike an effective balance between model complexity, computational demands, and the ability to generalize well, with a particular focus on addressing class imbalance issues and ensuring that our experimental findings could be reliably replicated.

Results

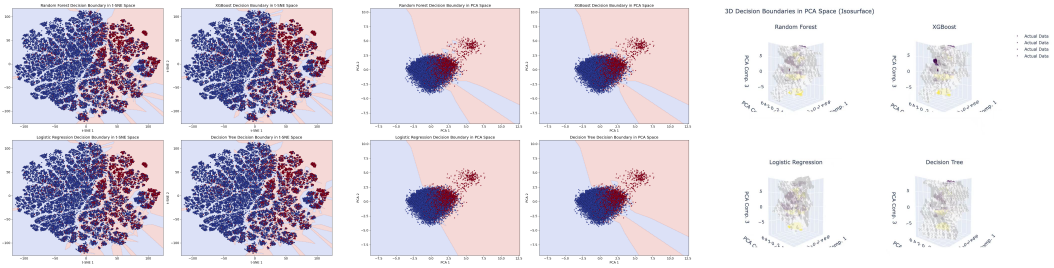
After training and testing the models, we can get the results, which is illustrated in Figure 3.

Figure 3: This is the performance comparison for the two vanilla models.



(a) Performance metrics comparison (b) Confusion matrix for logistic regression (c) Confusion matrix for decision tree

Figure 4: Comparison of 2D decision boundary visualizations using t-SNE and PCA. Specially, for PCA analysis, the feature which has the greatest absolute weight on PC1 is age (0.48158), while the one with greatest absolute weight on PC2 is education-num (0.673848).



(a) 2D t-SNE Decision Boundary (b) 2D PCA Decision Boundary (c) 3D PCA Decision Boundary

Following model training on the designated training data, the resultant decision boundaries were visualized, as depicted in Figure 4. Visualizing decision boundaries derived from models fitted

in a high-dimensional space presents a challenge when projecting to 2D or 3D. An initial attempt involved iterating through data points in the t-SNE embedded space, classifying them with our trained high-dimensional model, and color-coding accordingly to infer a boundary. However, due to the non-linear nature of t-SNE, which can significantly distort the original decision boundary geometry, this approach did not yield readily interpretable information.

Subsequently, alternative methods for decision boundary visualization were explored, and we are trying to use PCA-based visualization. Principal Component Analysis (PCA) was employed to visualize the decision boundary, leveraging its property as a linear transformation. While PCA may not fully capture the complexity of a non-linear boundary, it can reveal which principal components are most influential in class separation. PCA was first performed on the original dataset to obtain the transformation matrices. These matrices were then used to project an approximation of the decision boundary.

The visualizations indicate that the decision tree model possesses greater expressiveness, capable of defining multiple, distinct decision areas and boundaries. In contrast, the logistic regression model appears less adept at handling the complex data distribution, with its decision boundary often transecting groups of samples from different classes. Things are even clearer if we take a look at the 3D case, where though we cannot clearly see the shape of the boundaries, but we can see that the tree based models do take more volumes, splitting the whole 3D space more precisely.

Despite this, however, the overall performance of the initial decision tree model, with an accuracy around 0.809 on the test set, was not exceptional and fell short of the typical efficacy associated with tree-based models on complex datasets. Recognizing that the strength of tree models lies in their capacity to create flexible decision regions, we hypothesized that increasing tree depth or employing ensemble methods could yield improvements.

Accordingly, the depth of a single decision tree was increased, and further experiments were conducted. The results are shown in Table 3 and the visualization of the decision boundaries is in Figure 5, which indeed confirm our arguments as the decision boundaries get more complicated while the accuracy increases. Note that in the end, the decision boundary is overally complicated to fit the train data, which causes the problem of overfitting and the model performance no longer increases as the depth of decision trees increases.

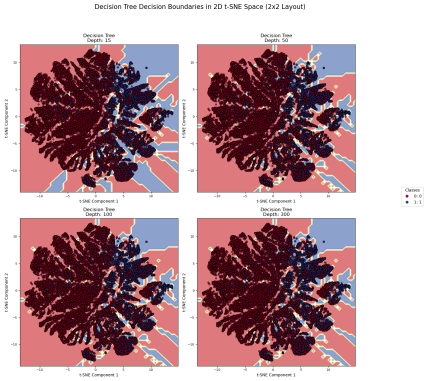


Figure 5: This figure shows the relationship between decision boundaries and depths of vanilla decision trees. Due to overfitting, the decision boundaries do not differ too much from depth between 50 to 300.

Concurrently, ensemble models (Random Forest and XGBoost) were also evaluated. The decision boundaries for these models are visualized in Figure 4, and their performance metrics are detailed in Figure 6. The results also show that the ensemble models demonstrate the capability to generate more complex and robust decision boundaries, contributing to more precise predictions. We also did experiments varying the depths of these ensemble models and their performance is in Table 3 (due to time constraints, we are unable to visualize the decision boundaries for the ensemble models with different depth). However, we may see that the performance is even worse than the vanilla model. After reading the lecture slides, we found that the best depth for ensemble models is usually less than 10, which means the hyperparameters we are currently using will potentially make the model overfit. More experiments need to be done to validate this argument.

Table 3: Comparison of the accuracy of different models at different depths

(a) The accuracy of vanilla decision tree in different depth

depth	train_acc	test_acc
15	0.860452	0.808913
50	0.998947	0.822835
100	0.998947	0.822630
300	0.998947	0.822630

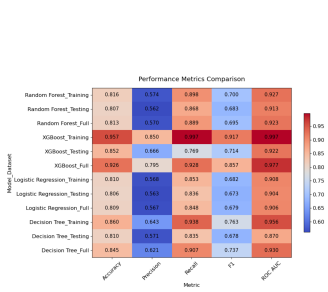
(b) The accuracy of Random Forest in different depth

depth	train_acc	test_acc
15	0.816052	0.806934
50	0.837404	0.821675
100	0.837287	0.821470
300	0.837287	0.821470

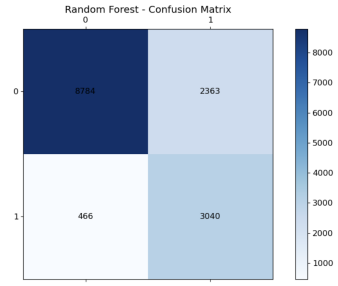
(c) The accuracy of XGBoost in different depth

depth	train_acc	test_acc
15	0.952449	0.849997
50	0.999591	0.852522
100	0.999591	0.852180
300	0.999591	0.852180

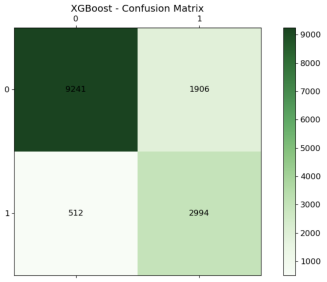
Figure 6: This is the performance comparison for the two ensemble models.



(a) Performance metrics Comparison (the heat map)



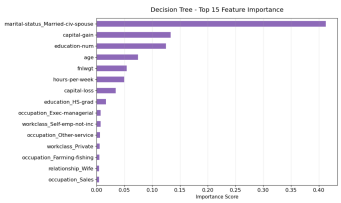
(b) Confusion matrix for random forest



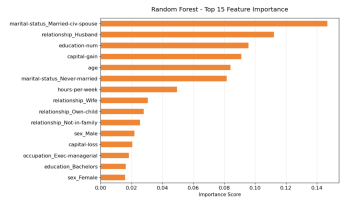
(c) Confusion matrix for XGBoost

Feature importance analysis, presented in Figure 7, provides additional support for our guess in Section 2.2, that only a few features are most important in separating the samples apart, and we can clearly see that whether getting married is the most important factor, which aligns to our common sense. And other features receive considerably less attention. An interesting observation is that the top performance model, XGBoost, mainly focus on one feature and in contrast, the standard decision tree and random forest models distribute importance more broadly across features, which correlates with their comparatively weaker performance. This observation aligns with the operational mechanisms of these algorithms; gradient boosting models like XGBoost iteratively learn from errors and are adept at identifying features that most effectively contribute to error reduction, whereas standard decision trees and Random Forest models may give more uniform consideration to all features, making them potentially more susceptible to noise.

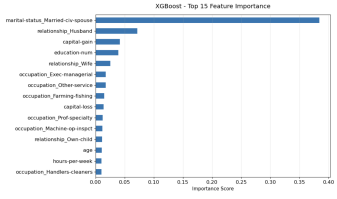
Figure 7: This shows the most important features in the dataset.



(a) Top features obtained by vanilla decision tree



(b) Top features obtained by random forest



(c) Top features obtained by XGBoost

It was also observed that model performance on one class was significantly poorer than on the other, potentially attributable to an imbalanced distribution of class labels in the training data. To address this problem, we investigate two resampling strategies, under-sampling and over-sampling. Under-sampling tries to select only a part of the samples from one class to reduce the model bias toward the majority and lowers the computational costs, while over-sampling will generate new data by taking pairs of samples from the minority class, and for numeric attributes, we'd interpolate a new data point between two existing ones; for non-numeric attributes, we would randomly choose

a value from one of the two parent samples for each feature. This will augment the representation of the minority class and improves the recall on high-income samples by giving the classifier more positive examples to learn from. Empirical results illustrated in Figure 8 show that both approaches effectively mitigate imblance and enhance overall robustness.

Figure 8: Analysis of label imbalance and its effect on accuracy. For the synthetic data strategy, we oversample to 26000 samples for the class with less samples. As for undersampling, we sample 8100 samples from each class (without replacement).

(a) Impacts of label imbalance on test set.

(b) Effects of undersampling on test set.

(c) Effect of oversampling on test set.

Model Selection

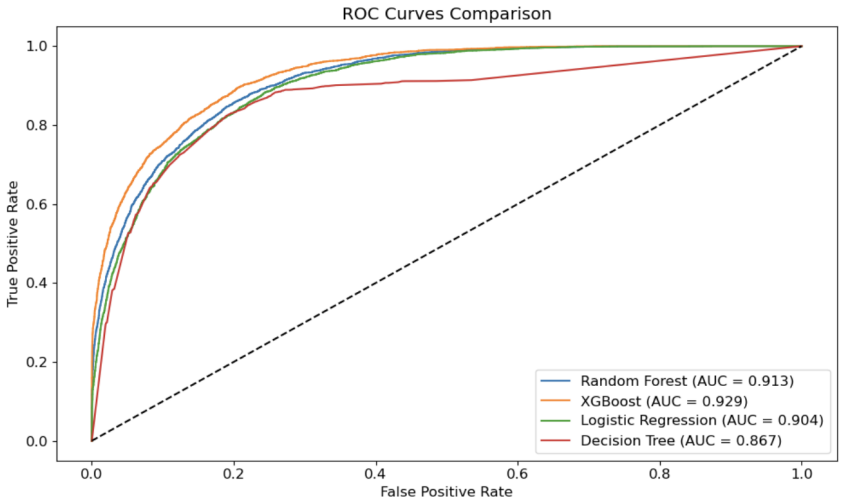


Figure 9: This figure shows the ROC curve and AUC value of each model.

The primary objective in this task is to identify a model class that effectively utilizes the available data while maintaining strong generalization capabilities on unseen data. Model performance was evaluated using the Receiver Operating Characteristic (ROC) curve and the corresponding Area Under the Curve (AUC) value, as illustrated in Figure 9. From these results, the XGBoost model, configured with a depth of 15 and 200 trees achieved the highest AUC value. Consequently, XGBoost is recommended for future classification tasks on this dataset.

3 Conclusion

In this work, we presented a comprehensive analysis of the Census Income dataset [1] for the task of binary income classification. Our pipeline began with rigorous data preprocessing—imputing missing values, one-hot encoding categorical variables, and standardizing numerical features—yielding a 105-dimensional feature space. Exploratory visualization via t-SNE revealed that (i) one-hot encoding

induces high dimensionality, (ii) the two income classes are not cleanly separable in a low-dimensional embedding, and (iii) sub-clusters exist within each class, pointing to feature redundancy.

To probe intrinsic structure, we applied both K-means and agglomerative hierarchical clustering. While K-means achieved superior internal cohesion and separation metrics, agglomerative clustering aligned more closely with true labels (as measured by ARI), indicating that income classes form complex, non-convex patterns unsuited to spherical cluster assumptions. Building on these insights, we evaluated logistic regression, decision tree, Random Forest, and XGBoost classifiers. Ensemble methods—particularly XGBoost—consistently outperformed simpler approaches by iteratively correcting errors and prioritizing the most informative features, achieving the highest AUC on held-out data.

We further addressed the pronounced class imbalance (37 155 instances labeled "$\leq \$50\,000$" vs. 11 687 instances labeled "$> \$50\,000$") through under-sampling of the majority class and synthetic over-sampling of the minority. Both strategies effectively mitigated bias toward the majority class and enhanced recall on high-income samples, demonstrating that careful sampling is critical for robust performance.

Our findings underscore three key lessons:

1. A substantial number of original features are redundant and should be pruned or re-engineered.
2. Non-linear, tree-based ensemble models such as XGBoost are well suited to the dataset's complex distribution.
3. Sampling techniques are indispensable for handling class imbalance in socio-economic data.

Future work will focus on advanced feature selection methods, exploration of alternative imbalance remedies (e.g., cost-sensitive learning), and extending the framework to other socio-economic datasets.

Credit

- **Hua XU**: Draft all the report and coordinate team members.
- **Jianhao RUAN**: Data visualization, organize the coding part and polish the report.
- **Leyi WU**: Responsible for data processing and model training and polish the report.
- **AI**: Formatting latex tables and figures, polishing the report (while all every single argument is made by us; this can be referred to with GitHub commit history), drafted the Introduction section; Q&A on the in-class contents.

References

[1] Barry Becker and Ronny Kohavi. Adult. UCI Machine Learning Repository, 1996. DOI: https://doi.org/10.24432/C5XW20.

Mainly the lecture slides. Many thanks to Prof. ZHONG and Prof. YANG.