Machine Learning Approaches for Income Prediction: Insights from the Census Income KDD Dataset

Joel Solé and Carlos Arbonés

GCED, UPC.

Abstract

This paper presents a study conducted on the Census Income Data Set to develop a machine learning model for predicting annual income levels based on demographic and employment-related factors. The data set comprises 200,000 rows and 42 columns, extracted from the 1994 and 1995 Current Population Surveys conducted by the U.S. Census Bureau. The main objective is to explore the relationships between various characteristics and income levels while identifying the most influential predictors. The paper describes the data preprocessing steps, including cleaning, transformation, feature selection, and model tuning techniques. The performance of the machine learning model is thoroughly evaluated using different metrics, and the importance of each variable in predicting income levels is interpreted. The results reveal key factors such as age, gender, weeks worked, and capital gains that significantly contribute to determining income. The study provides insights into the factors associated with higher or lower incomes and their implications for income inequality. This research has practical implications for policymakers, employers, and individuals interested in understanding income determinants, enabling targeted policies and informed decision-making.

Contents

1 Introduction

Predicting income is a crucial task in data science and machine learning that gives us insights into the socio-economic landscape. In this project, we dive into the analysis of the *Census-Income (KDD)* Data Set from the *UC Irvine Machine Learning Repository*. Our goal is to build a machine learning model that can accurately predict if someone makes more than 50,000$ a year based on various factors.

Our study aims to uncover what factors are linked to income differences and find out which predictors have the most impact on income. By using machine learning techniques, we hope to discover patterns and insights that help us understand income inequality better.

To achieve our goal, we follow a systematic approach. First, we carefully clean and process the Data Set to make sure the data is reliable and consistent. We take care of any missing values or errors to ensure the quality of our analysis.

Next, we choose and fine-tune a machine learning model that is suitable for our task. We experiment with different algorithms like logistic *regression, random forests,* or *boosting*. We evaluate their performance and select the model that gives us the most accurate and reliable predictions.

Furthermore, we dig into interpreting the developed model to understand the importance and influence of each factor. This analysis helps us gain valuable insights into the underlying reasons and key factors that contribute to high or low income levels.

2 Dataset Description and Characteristics

The *Census-Income (KDD)* dataset[1] is derived from the 1994 and 1995 current population surveys conducted by the U.S. Census Bureau. It contains weighted census data collected to predict income levels based on socio-demographic factors. The dataset was extracted by the *United States Department of Commerce* on March 7, 2000.

The dataset consists of a total of 199523 instances, each representing an individual observation. It includes 41 variables, such as *age*, *education*, *class_worker*, etc. and the target variable *income_50k*. (the complete list of variables can be found in Table 2)

The dataset encompasses a variety of variable types, including continuous, numeric, and categorical variables, providing a comprehensive view of socio-demographic characteristics.

It is important to note that the dataset contains missing values, which require appropriate handling during the data analysis process.

3 Data exploration and preprocessing

We begin by reading the data and dropping the *unknown*[1] column, as it should not be used for classifiers according to the dataset description. By removing this variable, we are left with 40 explanatory variables.

3.1 Basic inspection of the data

The first thing we observe when analyzing this dataset is that out of the 41 variables it contains, 29 are categorical and 12 are numeric (integer type). Additionally, we have a few numerical variables that should actually be treated as categorical, such as *det_ind_code* or *det_occ_code* that represent detailed industry and occupation codes respectively, as well as age and year, which we could treat as categorical variables.

[1] The *unknown* column in the dataset represents the *instance weight* attribute, which indicates the population representation per record in stratified sampling. While important for analysis, it should not be used for classifiers according to the UCI MLR (original source).

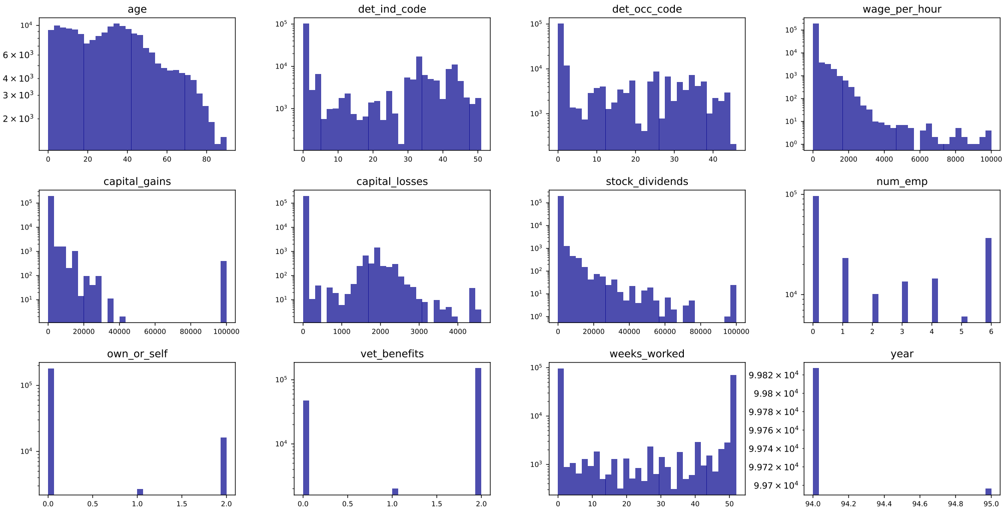


Fig. 1: *Histogram for all the numerical explanatory variables in logarithmic scale*

Another thing we observe about the numerical variables is that, apart from *age* and *year* (which can not be really treated as numerical), the rest are highly unbalanced, as they have the vast majority of values at 0. Furthermore, with the Figure 2 we observe that the categorical data is also unbalanced, as more than half of the features have a category that contains more than 80% of the data.

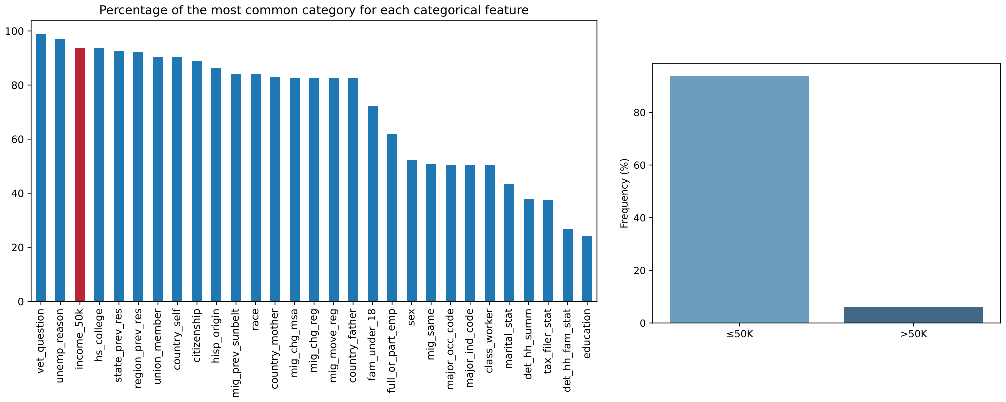


Fig. 2: *Categorical data analysis. The left histogram displays the percentage of the most common category for each categorical feature, with the target value plotted in red. The right bar plot shows the frequencies of the target variable income_50k.*

As we can see in Figure 2, we also have a significant imbalance in our target variable. This can lead to the model being biased towards the majority class, resulting in lower accuracy and predictive performance for the minority class. Imbalanced target variables pose a challenge in machine learning as the model tends to prioritize the dominant class during training, resulting in limited learning and misclassification of minority class instances. To address this issue, various techniques can be employed, such as oversampling the minority class, or undersampling the majority class. These techniques aim to rebalance the dataset and provide the model with sufficient exposure to the minority class, enabling it to learn accurate representations and improve performance on both majority and minority classes. In our case, due to the large amount of data available, we have chosen to perform undersampling [2], since it can be computationally less costly and allows us to reduce the number of samples, thus speeding up the training of our models.

Data cleaning

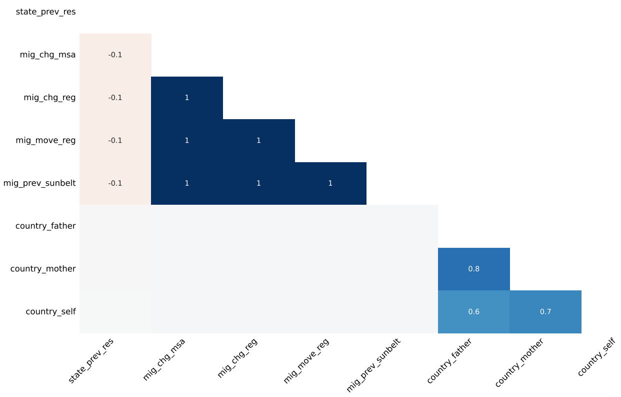
As a first step, we have removed duplicates from the dataset to ensure data cleanliness and consistency, since having duplicate data in a dataset can lead to biased and inaccurate results. This is because duplicate data can skew the distribution of values and lead to over-representation of certain data points. Additionally, it can result in inefficiencies in data processing and model training, as the same information is being analyzed multiple times.

3.2 Missing values treatment

In this section, we will examine the occurrence of missing values within the dataset and outline the appropriate strategies for addressing each case. The handling of missing data is a crucial step in the data preprocessing phase, as it can have a significant impact on the quality and reliability of the analysis and modeling results. Therefore, it is essential to carefully evaluate and handle missing data appropriately to avoid any biases or errors in the downstream analyses.

In the figure 3, we observe that we have 4 columns that have 49% of missing values: *mig_chg_msa, mig_chg_reg, mig_move_reg,* and *mig_prev_sunbelt*, and that the correlation between these 4 features is 1. This correlation is due to the fact that, for the 4 columns, we have 100% of missing values in the year 1995, while none in the year 1994. Since the data comes from surveys, we can assume that the migration-related questions ceased to be asked in 1995.

On the other hand, we have that the missing values in *country_father, country_mother,* and *country_self* also have a certain correlation, although it is not as high as the 4 previous features.



Column	Missing values
state_prev_res	0.46%
mig_chg_msa	49.02%
mig_chg_reg	49.02%
mig_move_reg	49.02%
mig_prev_sunbelt	49.02%
country_father	4.17%
country_mother	3.80%
country_self	2.17%

Fig. 3: *Distribution of missing values in the dataset. The left plot displays a heatmap illustrating the correlations between the missing values. The right table provides the percentage of missing values for specific columns in the dataset.*

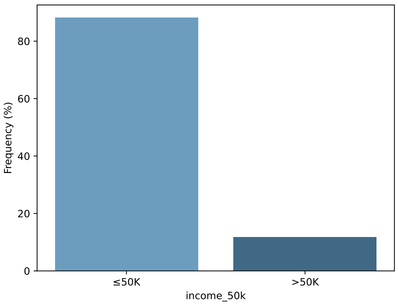
[2] In undersampling, a subset of the majority class instances is randomly selected or strategically removed until the desired class balance is achieved. The implementation was done using the imbalanced-learn python library[2]

For the missing values of the migration-related features, we have removed the entire columns since they have a high percentage of missing values (49%), and therefore do not have enough meaningful data to be considered significant. Additionally, imputation does not make sense in this case due to the large number of missing values.

For the other variables with missing values, which account for less than 5% of the data in all cases, we will perform imputation. Since all the remaining missing values come from categorical variables, we decided to use the most frequent value of the respective variable to impute them. Note that imputation will be performed separately on the training and test data to avoid data leakage and maintain indepence on the test data.

3.3 Outlier treatment

For this section, we will focus on the variables *capital_gains*, *capital_losses*, *wage_per_hour*, and *stock_dividends*, as they exhibit some unusual values at the extremes. As seen in Figure 4, the maximum values of *capital_gains*, *wage_per_hour*, and *stock_dividends* appear to be artificial. Additionally, we would expect a significant portion of individuals with *capital_gains* of 99999 to earn more than 50k annually. However, as observed in the left plot of Figure 4, this is not the case.



Feature	**Max value**	**Count**
capital_gains	99999	390
capital_losses	4708	4
wage_per_hour	9999	1
stock_dividends	99999	25

Fig. 4: *The left plot displays the distribution of the target value for the individuals with a capital gain of 99999. The right table shows the maximum value of four features, as well as the count of the individuals with the maximum value for each feature*

Hence, we can consider these artificial values as outliers, and since they do not represent a large portion of the overall data, we can remove them.

The remaining numerical values do not appear to exhibit any outliers. However, we have noticed some peculiarities, such as individuals with an age of 0. Since these instances have consistent values [3], indicating that they represent newborns, we have not considered them as outliers.

3.4 Categorical conversion

In this section, we will convert some numerical variables that we find convenient into categorical variables. We will begin by transforming numeric variables that are in fact categorical, such as *det_ind_code* and *det_occ_code,* which respectively represent industry and occupation codes, and also *own_or_self, vet_benefits,* and *year*, which can be directly converted into categorical variables since they have between 2 and 3 categories only.

To convert *age* into a categorical variable, we generate 10 bins: from 0 to 18, from 18 to 25, and in 10-year intervals up to 105. We can observe the resulting categorical features after we converted them in the figure 10.

[3] all individuals with an age of 0 have annual incomes below 50k and zeros in economic variables such as hourly wages, capital gains, number of employees, etc.

The remaining numerical features in the dataset can be regarded as numerical variables since they include economic variables such as *wage_per_hour, capital_gains, capital_losses,* and *stock_dividends,* as well as count variables such as *num_emp* and *weeks_worked*. However, it is worth noting that the remaining numerical variables exhibit a high degree of skewness, as depicted in Figure 11, with a substantial number of instances being exactly zero. As a result, no transformation method can be applied to normalize them.

Note that, most of models we fit do not accept categorical variables directly, so we will need to encode them using one-hot encoding [4].

3.5 Feature extraction

After data cleaning, we are left with the numerical variables as shown in Figure 11. One approach to reduce the dimensionality of the data is to combine related columns. In this regard, we can generate a new column called *capital_balance* by subtracting *capital_losses* from *capital_gains*. We can observe the resulting new feature in Figure 5.

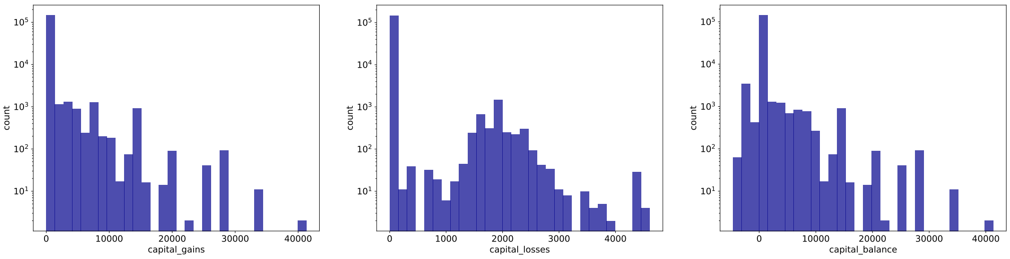


Fig. 5: *Histogram in logarithmic scale of capital_gains, capital_losses, and capital_balance*

3.6 Preprocessing methodology

As we have seen in previous sections, we have different ways of preprocessing the data that can be more effective depending on the model we are going to fit. Therefore, we have decided to encode the different data preprocessing methods as if they were hyper-parameters, in order to find the best preprocessing approach for each of the models tested. All possible parameter combinations can be found in Table 3, but please note that we will not test every single combination. Instead, depending on the model we are fitting, we will focus only on the combinations that make the most sense to try for that specific model. The details of how we will select these parameters for each model are explained further in the *parameter tuning* section.

[4] One-hot encoding is a technique used in machine learning to convert categorical variables into a binary representation. It creates binary columns for each category, with a value of 1 indicating the presence of that category and 0 otherwise.

4 Modeling

After preprocessing the data, we can proceed with model fitting. Since our problem involves binary classification, we will fit eight different models and select the one that achieves the best metric. The majority of models were fitted using the *scikit-learn* [3] Python library, with the exception of *CatBoostClassifier* [4].

The theoretical foundations of the models, including the assumptions and the step-by-step procedure, are based on [5], [6], and [7].

In the following section, we will discuss in more depth the proposed models, and outline the procedures behind them. Additional information on the models tested will be included in the appendix[5].

4.1 Methodology

The first step is to define how we evaluate our models and which metric to use. As discussed in previous sections, the initial data is highly imbalanced. Therefore, accuracy would not be a suitable metric, as even the simplest model that always predicts $\leq$*50k* would achieve 93.8% accuracy. Hence, we need to consider another metric. Given our objective of balancing both false positives and false negatives, we have decided to utilize the f1-score[6].

Once we have determined the metric to choose the models, we need to decide how to evaluate them. Firstly, we have divided the data into training and test sets, with 70% and 30% of the data, respectively. The test set will only be used to evaluate the generalization metrics of the final model and its prediction capability on independent data. Therefore, model adjustment and evaluation will be performed solely on the training set. In order to compare the models, it is important to assess their generalization capabilities over a new data set that the model has not been trained on. Hence, we have chosen to evaluate the f1-score of each model using cross-validation.

Note that in each split of the cross-validation, preprocessing is applied separately to the training and validation folds. This is necessary because scaling the data or performing imputation requires using the remaining data within the fold. Similarly, actions involving row elimination, such as removing duplicate data or downsampling, will only be applied to the training fold.

Parameter tuning

An essential aspect of model fitting is the identification of optimal hyperparameters. In our case, since we have different ways of preprocessing the data, we aim to find, for each model, the best data preprocessing approach (among those mentioned in previous sections), along with the optimal set of model hyperparameters.

For this optimization process, we also employ cross-validation as it allows us to identify parameters that lead to better generalization performance by the model. One approach to finding the best parameters would involve fitting all possible combinations of preprocessing and model parameters. However, the number of potential combinations increases exponentially with the parameters being tested. Hence, we initially select default model parameters and use them to find the best preprocessing parameters [7]. Likewise, utilizing the identified preprocessing parameters, we then search for the optimal model parameters. Subsequently, we retain only the top 15 parameters for both preprocessing and model and repeat the process until we reach an iteration where the

[5] A wide variety of models were tested, although not all of them were initially considered suitable for our dataset. This will be further discussed in the description of each model.

[6] The formula for the f1-score is given by $\frac{TP}{TP+\frac{1}{2}(FP+FN)}$

[7] To identify the best preprocessing or model parameters, we do search among all possible combinations, as the individual combinations for each component are significantly fewer than the joint combinations

desired metric, in our case, the f1-score, does not show further improvement. Note that for each of the tested preprocessing and model parameter combination, we used 5-fold cross-validation to obtain the f1-score.

Through this iterative approach, we have been able to discover the preprocessing and model parameters that yield improved f1-scores for each of the fitted models. The determined optimal preprocessing parameters through this approach are presented in Table 4, while the optimal model parameters are outlined in Table 5.

4.2 Discriminant Analysis

Discrimination methods are based on assigning to a new sample the category that minimizes the expected loss. It can be shown that minimizing the expected loss is equivalent to assign to new sample the category that maximizes $Pr(G = k|X = x)$. Using Bayes' theorem we get to the following probability:

$$Pr(G = k|X = x) = \frac{Pr(X = x|G = k) \cdot Pr(G = k)}{Pr(X = x)} = \frac{f_k(x) \cdot \pi_k}{\sum_{l=1}^{K} f_l(x)\pi_l}$$

Here, $f_k(X)$ represents the class-conditional density of X given the class $G = k$ (which is assumed to follow a multivariate normal distribution in discriminant analysis), and π_k represents the prior probability of class k. Note that $Pr(X = x)$ is constant for all categories, so we will choose the one that maximizes $f_k(x)\pi_k$.

Depending on whether the covariance matrices for different classes are assumed to be equal or not, two derivations and two discriminant functions can be obtained. In LDA, we assume a common covariance matrix $\Sigma_k = \Sigma\ \forall k$, while in QDA, Σ_k is not assumed to be equal. In the first case, with convenient cancellations, we arrive at the discriminant function for LDA:

$$\delta_k(x) = x^T \Sigma^{-1} \mu_k - \frac{1}{2} \mu_k^T \Sigma \mu_k + \log \pi_k$$

While in QDA, these cancellations do not occur, and particularly the quadratic terms in x remain. We obtain the quadratic discriminant function for QDA:

$$\delta_k(x) = -\frac{1}{2} \log |\Sigma_k| - \frac{1}{2}(x - \mu_k)^T \Sigma_k^{-1} (x - \mu_k) + \log \pi_k$$

Although our dataset does not fully meet the assumptions of the model (discussed in Appendix 9.3.1), we still attempted to apply discriminant analysis due to its suitability for our purpose. It is worth noting that the covariance matrices were not significantly equal, so in this case, we should apply QDA. However, LDA, being a method with fewer parameters, could perform better on the test set and is much faster due to its linearity. This will be discussed further in the Results Comparison section.

Discriminant analysis is a rapid and simple method for creating separation between groups. Considering the large number of samples we have, it was a practical choice.

4.3 Support Vector Machines

The goal of Support Vector Machines (SVM) is to find a hyperplane that optimally separates the data, maximizing the distance to any data point. This distance is known as the classifier's *margin*. A classifier with a larger margin tends to make fewer misclassifications, as the points farther from the hyperplane provide greater confidence in their classification.

In our case, we will utilize the primal formulation of SVM with soft margins instead of the dual formulation. The dual formulation is not computationally feasible due to the large number of samples, so no kernel trick can be applied either.

The primal formulation is expressed as follows:

$$\begin{aligned} \min_{\mathbf{w},\gamma,\mathbf{s}\in\mathbb{R}^{n+1+m}} \quad & \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{m}\mathbf{s}_i & \\ \text{s.t.} \quad & y_i(\mathbf{w}^T\mathbf{x}_i + \gamma) + \mathbf{s}_i \geq 1 & i = 1,\dots,m \\ & \mathbf{s}_i \geq 0 & i = 1,\dots,m \end{aligned}$$

SVM was chosen for our dataset due to its effectiveness in handling high-dimensional data and its memory efficiency. It is known to perform well in scenarios with a large number of features like in our dataset.

However, it is important to acknowledge the limitations of SVM. It may not be suitable for large datasets, and since our dataset contains a significant number of samples, training SVM can be time-consuming. Additionally, SVM does not provide a direct probabilistic explanation for classification, and it does not inherently provide knowledge of the likelihood of a sample belonging to a particular group.

4.4 Logistic Regression

Logistic regression is a linear discriminative method that aims to model the posterior probabilities of classes, which represent the probability of an observation belonging to a particular category. Unlike other methods such as the *perceptron* or *SVM*, logistic regression provides information about the likelihood of our prediction.

For this reason, logistic regression makes use of the *sigmoid* function, which allows us to interpret the results as probabilities and reflect the uncertainty. We assign observations to a group based on whether $\sigma(w^T x) > 0.5$, where σ is the *sigmoid* function. If the value is greater than 0.5, the observation is assigned to one group, and if it is less than that value, it is assigned to the other group. The values of w are obtained through likelihood estimation.

We chose to implement logistic regression in our dataset due to its simplicity, interpretability, and computational efficiency. Logistic regression is easy to understand, implement, and train, making it a convenient choice. It does not assume a specific class distribution, making it suitable for various data scenarios like our dataset. Is also fast in classifying unknown records and can easily accommodate new data points. However, our dataset does not fully fulfill the assumptions of the model (discussion made in appendix).

4.5 K-Nearest Neighbors

The k-NN algorithm predicts the class of a sample by considering the combinations of targets from the k closest training examples. In our case, we will utilize the majority vote from these k closest samples to determine the predicted class label.

We chose the k-NN algorithm based on its ease of understanding, implementation, and explanation. As a non-parametric algorithm, it does not have strict assumptions. Furthermore, k-NN does not require training steps, utilizing the training data at runtime for predictions, resulting in faster processing compared to other algorithms.

One of the main assumptions of the k-NN algorithm is that the distance between samples can be calculated. In our case, we addressed this assumption by utilizing the *Minkowski distance* metric (with $p = 2$). This choice was necessary since we applied one-hot encoding to encode categorical variables, allowing us to calculate distances between samples in the feature space.

Some limitations of the method in our dataset is that k-NN may be inefficient and slow when dealing with large datasets due to the high computational cost of calculating distances between new points and the training data. Additionally, k-NN does not perform well with high-dimensional data, as finding distances in higher dimensions becomes more challenging. Moreover, it is sensitive to outliers, as they can significantly impact the algorithm's performance.

4.6 Random Forest

The idea behind tree-based methods is to partition the feature space into a set of rectangles and then fit a simple model within each of them. In our dataset, we could try fitting a classification tree. However, instead, we have opted to use a random forest. By using *bootstrap aggregation*, we aim to reduce the variance of the predictions.

The bootstrapped dataset is created by randomly selecting samples from the original dataset with repetitions allowed. The samples that are not selected for the bootstrapped dataset are referred to as *out-of-bag* (OOB) samples[8].

We construct a large collection of unrelated trees and then gather the majority of votes from them (the step-by-step implementation can be seen in Figure 14). This ensemble approach helps improve the overall predictive accuracy and generalization of the model.

The hyperparameters to be estimated through cross-validation are listed in Table 1. The *n_estimators* hyperparameter refers to the number of trees in the random forest. In the context of machine learning, a commonly accepted value is 100, but we will try different values to find the optimal one. The criterion hyperparameter determines the quality measurement function used for splitting. We can assess the quality of a split by using either gini or entropy, which are similar metrics that measure the purity of the split.

Table 1: *Hyperparameters of random forest*

Parameter	Values
n_estimators	[50, 75, 100,125]
criterion	['gini']
max_depth	[None, 25, 30, 35, 40]
max_features	['sqrt', 'log2']
class_weight	[None, 'balanced']

The *max_depth* hyperparameter represents the maximum depth of each tree, and we have experimented with various values. The *max_features* hyperparameter, denoted as m in the random forest algorithm (see Figure 14, step *(b) ii)* , determines the number of randomly selected features used for each split. For classification tasks, it is common to choose $\lfloor\sqrt{p}\rfloor$, where p is the total number of features. However, we have also explored the option of using $\log_2(p)$ as an alternative value.

Finally, the value of *class_weight* determines whether we want to give more importance to the classifications of the class that has less frequency. This is useful for imbalanced datasets like ours, where the weights of the samples belonging to minority class (*income_50k = 1*) will be increased. If *class_weight* is set to *None*, all classes have the same weight.

Variable Importance

A crucial aspect of random forests, and one of the reasons why we have chosen this method for our dataset, is the ability to estimate the importance of variables in the

[8] Approximately $\frac{1}{3}$ of the dataset is not selected for the bootstrapped dataset, and these samples can be used later to estimate the generalization error (without the need for cross-validation) and study variable importance.

model. By ordering the variables based on their importance, we can gain insights into which variables have the greatest impact on income levels. This measure of importance is similar to that of gradient boosting methods, which will be discussed later in Feature Importance. Two ways to calculate variable importance will be studied, one based on Gini and the other one based on Random variable permutation.

4.7 Gradient Boosting

Finally, we decided to fit two different models based on gradient boosting methods. Gradient boosting is a machine learning technique that combines multiple weak models to create a strong predictive model. In our fitted models, decision trees with small depth were used as weak models. Then, it iteratively builds an ensemble of models, with each one focusing on correcting the mistakes made by its predecessors. The models are trained to predict the residuals of the previous models by optimizing a loss function using gradient descent. The predictions of all the models in the ensemble are then combined to form the final prediction. The detailed explanation of the algorithm in pseudo code can be found in Figure 15.

In our case, we have decided to fit one model using XGBoost and another using CatBoost. We chose to use XGBoost because it is one of the most popular libraries for gradient boosting. One limitation it has is that it does not handle categorical variables directly, so one-hot encoding was necessary to convert the categorical variables into a suitable numerical format.

On the other hand, CatBoost[8] is a library that accepts categorical variables and internally applies more advanced encodings, such as *ordered target encoding*. Since our dataset contains many categorical variables, the features provided by CatBoost proved to be very useful, as we will see in subsequent results. CatBoost's ability to handle categorical variables effectively contributes to capturing the nuances and patterns within the data, potentially leading to improved model performance.

Moreover, both XGBoost and CatBoost are based on an ensemble of decision trees. Decision trees are inherently interpretable models, allowing us to extract insights and interpret the results. One useful feature provided by these models is the ability to compute feature importances, which indicates the relative importance of each feature in the model's decision-making process (the discussion is made in Feature Importance). Understanding feature importances can help identify the most influential variables in predicting the target variable and provide valuable insights into the underlying relationships in the data.

4.8 Model Results Comparison

Finally, after finding the best parameters for each model and preprocessing technique mentioned, we conducted a final 20-fold cross-validation to obtain a more precise estimation of the generalization performance. In this section, we will compare the results obtained by each method that can be found in Figure 6.

Upon examining the results, the first thing we observe is that the methods with comparatively worse metrics are LDA, KNN, and QDA, with QDA performing the worst.

On one hand, we anticipated that the discriminant analysis methods would not yield very good metrics because we do not satisfy the assumption of normality, and we have no evidence that the covariance matrices are the same for both clusters when using LDA. On the other hand, QDA needs to estimate $\frac{1}{2}p(p+3)+1$ parameters, where p is the dimension of the features. As we perform one-hot encoding on the categorical variables, the dimension is $p = 474$, so QDA needs to estimate around 113 thousand parameters, which is more parameters than we use to fit the model after downsampling (hence the optimal downsampling found in Table 4 is very low with a target frequency of 0.90). This leads to severe overfitting, resulting in poor metrics for QDA.

Furthermore, the large number of categorical variables also affects KNN. With one-hot encoding, apart from significantly increasing the dimension, most of the features have binary values, making it ineffective to calculate distances between individuals.

On the other hand, we see that SVM and Logistic Regression show better results than the previous three models, but they still do not provide the best results. Since we couldn't use the kernel trick for SVM, both models employ linear decision boundaries, which do not capture complex relationships between the data effectively.

Finally, we observe that the models offering the best results are ensemble models based on decision trees. These models allow for capturing nonlinear relationships, which work well for our dataset. Moreover, the models that perform the best are Boosting models, as they work effectively for large datasets like ours and are robust against overfitting.

5 Model Selection

Among the Boosting models tested, we observe that the model with the highest f1-score is Catboost, with a value of 0.8. The superior performance of tree-based models can be attributed to their ability to capture complex relationships and interactions among variables, as well as their capacity to handle both numerical and categorical features effectively.

The high f1-scores obtained by these models indicate their effectiveness in accurately predicting income levels and capturing the underlying patterns in the dataset.

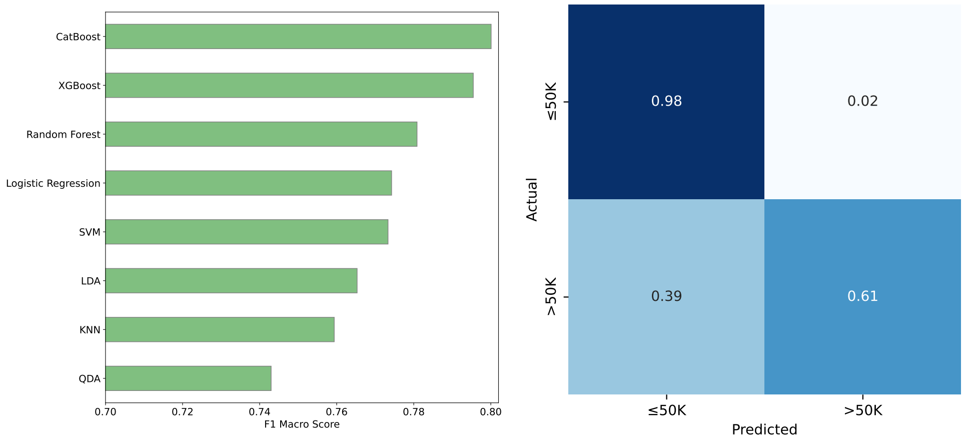


Fig. 6: *The plot on the left shows the models ranked by their f1-score, while the plot on the right displays the final confusion matrix of CatBoost on the test dataset.*

5.1 Prediction capability

Therefore, although the difference may not be significant, the model that we consider most suitable for our dataset based on the results is *CatBoost*. Once the final model is selected, the only remaining step is to estimate the model's generalization error on independent data, in this case, the test data that was set aside at the beginning.

We can observe that the resulting confusion matrix is similar to the one obtained earlier using cross-validation. The model consistently classifies individuals with annual incomes below 50k correctly, while it achieves a 60% accuracy for individuals with incomes above 50k. The model obtains an accuracy of 95% and an F1-score of 80% on the test dataset, indicating that it generalizes well to new data. Therefore, we can conclude that the model performs well and demonstrates good generalization capabilities.

6 Feature importance

In this section, we will examine the importance of the most significant variables in the model and draw conclusions that help us understand the underlying factors affecting income levels. In addition to evaluating the importance of variables, it is important to consider the correlation of these variables with the target variable. This will help us understand whether the variables that have significant weight in our model have a positive or negative impact on individuals' income.

As mentioned earlier, tree-based models allow us to extract information about which variables have the greatest impact on classification within each group. In this case, we will analyze and compare the results of variable importance analysis in both *random forest* and *gradient boosting* models.

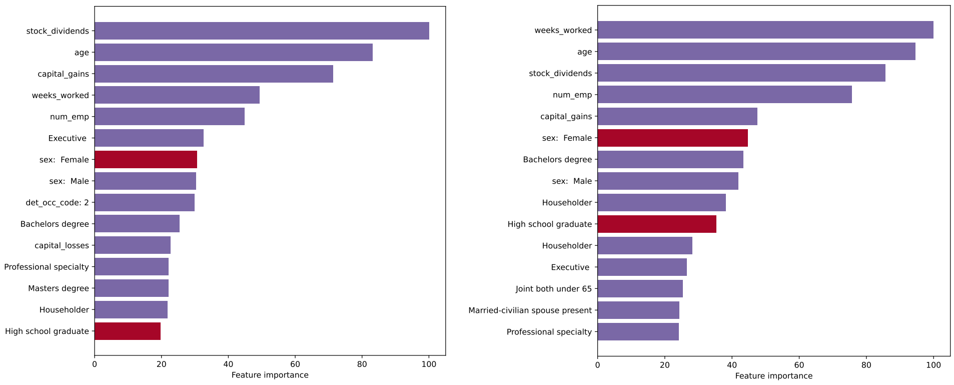


Fig. 7: *Variable importance plots for a classification* ***random forest*** *grown in the Census-KDD data. The left plot bases the importance on the Gini splitting index. The right plot uses OOB randomization to compute variable importance's. In red the features that are negatively correlated with the target variable*

As we can see in Figure 7, the most important variables appear to coincide with both metrics, as the top 5 variables in both rankings are the same. However, the order differs. Based on the Gini index, the most influential variables are *stock_dividents*, *age*, and *capital_gains*, all of which have a positive correlation with the target variable. This means that these three variables are important for predicting an income of over 50k. Based on OOB randomization, the four most important variables are *weeks_worked*, *age*, *stock_dividents*, and *num_emp*, all of which also have a positive correlation with the target.

Among the top 15 most important variables in the dataset, we observe that only 2 have a negative correlation, and both variables appear in both plots. One is *sex: Female*, and the other is *high school graduate*.

As we can observe in Figure 8, the variables that are considered important by *catboost* are very similar to those of *random forest*. With the Gini index, we see that *age*

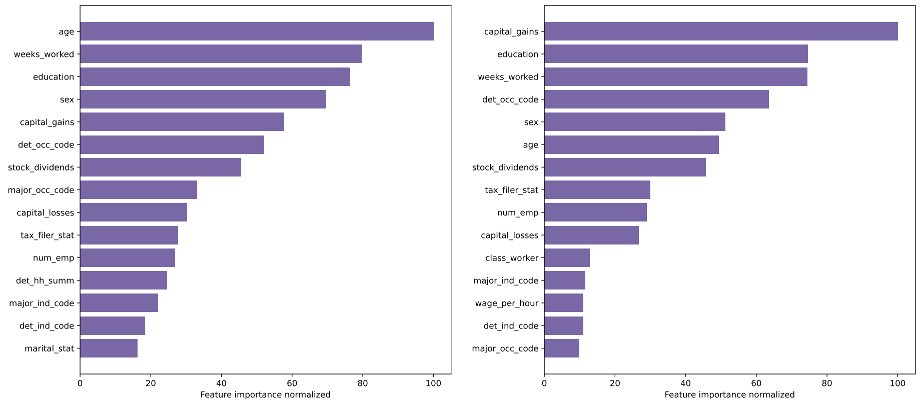


Fig. 8: *Variable importance plots for a classification* ***catboost*** *in the Census-KDD data. The left plot bases the importance on the Gini splitting index. The right plot uses OOB randomization to compute variable importance's.*

becomes the most influential variable, while with OOB randomization, it is *capital_gains* that has the most influence. We also see that *weeks_worked* is highly important and appears near the top in both plots. Variables such as *education* and *sex* also appear higher up, and we observe that *det_occ_code* (occupation code) is important as well.

7 Final Conclusion and Results

After obtaining the results from the different methods and studying which variables are most important for the target variable *income_50k*, we can draw some significant conclusions about the factors that are most important in an individual's economy.

There are no surprising variables that have emerged in the results, as the most important variables were quite obvious beforehand in terms of their relevance to determining whether a person earns more or less than 50k. Variables such as *stock_dividents* or *capital_gains* have a direct impact on a person's income and, therefore, have a positive correlation.

Additionally, *age* is important. It is uncommon for individuals under the age of 25 to earn more than 50k, which is why age has a positive correlation. *weeks_worked* also makes sense as an important variable, as the more weeks worked, the higher the likelihood of earning more money. We also see that *education* carries significant weight. Similar conclusions can be drawn for *num_emp* and *det_occ_code*.

Furthermore, there are two variables that have non-negligible importance: *Female sex* and *High School graduate*.

In Figure 9, we can see that around 2.5% of women earn more than 50k, while 10% of men earn above this threshold. This explains the negative correlation of the *female* category.

We can also provide an explanation for the negative correlation of *high school graduate*. In Table 6, we can observe the education levels ordered from highest to lowest academic relevance, along with the number of individuals in the dataset who have each level of education. *High school graduate* can be seen as equivalent to completing secondary education, and we notice that it is the most common education category among individuals. A negative correlation indicates that the majority of individuals with only this level of education tend to earn less than 50k. We can conclude that, in

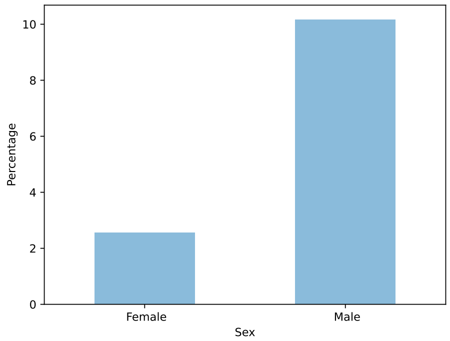


Fig. 9: *Percentage of people that earn more than 50k divided by sex*

general, having a university degree, master's degree, Ph.D., etc., contributes positively to income, as it shows a positive correlation.

In summary, the application of machine learning models in this study has provided valuable insights into the factors that influence an individual's income level. By examining the importance of various variables, we have gained a deeper understanding of the relationships and correlations between these factors and the likelihood of earning above or below 50k.

These insights can have significant practical implications. Firstly, they can assist individuals in making informed decisions regarding their career choices, education, and financial planning. Understanding which variables have the most significant impact on income allows individuals to prioritize areas of focus that can potentially lead to higher earnings. For example, the findings suggest that investing in higher education may contribute positively to income.

Furthermore, these insights can inform policymakers and organizations in designing targeted interventions and policies aimed at reducing income disparities. By identifying the variables that are most strongly associated with higher income levels, policymakers can develop strategies to support individuals in enhancing those specific factors.

8 Limitations and Prospects for Further Development

One of the major challenges of this dataset is the large number of variables, especially the fact that the majority of them are categorical. Handling categorical data is not a trivial task in machine learning, and there are many different ways to encode categories. In our case, we used one-hot encoding for models that do not accept categorical variables. However, since our dataset not only has many categorical variables but also each one has a large number of categories, this significantly increases the dimensionality of the data, approaching around 500 features when using one-hot encoding.

On the other hand, the good results obtained by CatBoost are partly due to its more advanced treatment of categorical variables. CatBoost internally applies more sophisticated encodings such as ordered target encoding, which can capture more nuanced information from the categorical variables.

Therefore, for future development, it would be interesting to explore other encoding methods such as Bayesian target encoding[9] or other approaches that could potentially handle the high-dimensional categorical data more effectively and potentially improve the other tested model's performance.

Another limitation to consider is that the data comes from surveys. This introduces a certain bias in the data itself since it does not reflect the entire U.S. population but rather the fraction of the population that agreed to respond to the survey. For example, it is possible that individuals with higher incomes are less likely to disclose their actual income, so if this is the case, the data would represent individuals earning less than 50k annually more accurately than those earning more than 50k.

References

[1] D. Dua, C. Graff. OpenML dataset: Dataset name. https://www.openml.org/search?type=data&sort=runs&id=42750&status=active (2019). Irvine, CA: University of California, School of Information and Computer Science

[2] Imblearn. imbalanced-learn - python reference. https://imbalanced-learn.org/stable/ (2023). [Online; accessed 30-May-2023]

[3] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, E. Duchesnay, Scikit-learn: Machine learning in Python. Journal of Machine Learning Research **12**, 2825–2830 (2011)

[4] CatBoost. CatBoostClassifier - python reference. https://catboost.ai/en/docs/concepts/python-reference_catboostclassifier (2023). [Online; accessed 30-May-2023]

[5] T. Hastie, R. Tibshirani, J. Friedman, *The Elements of Statistical Learning*. Springer Series in Statistics (Springer New York Inc., New York, NY, USA, 2001)

[6] M. Arias Vicente. Class lectures of aprenentatge automatic 1. Presentation (2023)

[7] KDnuggets. Machine learning assumptions: Part i. https://www.kdnuggets.com/2021/02/machine-learning-assumptions.html (2021). [Online; accessed 30-May-2023]

[8] A.V. Dorogush, V. Ershov, A. Gulin, Catboost: gradient boosting with categorical features support. arXiv preprint arXiv:1810.11363 (2018)

[9] M. Larionov, Sampling techniques in bayesian target encoding. arXiv preprint arXiv:2006.01317 (2020)

9 Appendix

The appendix contains additional material that complements the main body of the work, providing further support for visualization and understanding of the processes involved. It is divided into different sections, including Tables, Plots, and Supplementary Content.

9.1 Tables

Table 2: *Detailed column description*

Column Name	Description
age	Age of the worker
class_worker	Class of worker
det_ind_code	Industry code
det_occ_code	Occupation code
education	Level of education
wage_per_hour	Wage per hour
hs_college	Enrolled in educational institution last week
marital_stat	Marital status
major_ind_code	Major industry code
major_occ_code	Major occupation code
race	Race
hisp_origin	Hispanic origin
sex	Sex
union_member	Member of a labor union
unemp_reason	Reason for unemployment
full_or_part_emp	Full- or part-time employment status
capital_gains	Capital gains
capital_losses	Capital losses
stock_dividends	Dividends from stocks
tax_filer_stat	Tax filer status
region_prev_res	Region of previous residence
state_prev_res	State of previous residence
det_hh_fam_stat	Detailed household and family status
det_hh_summ	Detailed household summary in household
mig_chg_msa	Migration code - change in MSA
mig_chg_reg	Migration code - change in region
mig_move_reg	Migration code - move within region
mig_same	Live in this house one year ago
mig_prev_sunbelt	Migration - previous residence in sunbelt
num_emp	Number of persons that worked for employer
fam_under_18	Family members under 18
country_father	Country of birth father
country_mother	Country of birth mother
country_self	Country of birth
citizenship	Citizenship
own_or_self	Own business or self-employed?
vet_question	Fill included questionnaire for Veterans Admini...
vet_benefits	Veterans benefits
weeks_worked	Weeks worked in the year
year	Year of survey
income_50k	Income less than or greater than $50,000
edu_year	Number of years of education

Parameter name	options
imputation	{*mode*, *dropna*, *nacat*}
remove duplicates	{True, False}
scaling	{*minmax*, *standard*, None}
categorical age	{True, False}
remove outliers	{True, False}
merge capital	{True, False}
one hot encoding	{True, False}
downsampling method	{*Random*, *NearMiss*, None}
downsampling frequency	[0, 1]

Table 3: *Preprocessing hyperparameters (the underlined options are the ones that eliminate rows, thus cannot be used for the validation folds)*

Parameter	**lda**	**qda**	**knn**	**logreg**	**svm**	**rf**	**xgb**	**catboost**
scaling	None	None	standard	standard	None	None	None	None
cat_age	True	True	False	True	True	False	False	False
remove_outliers	True	False	False	False	True	False	True	False
merge_capital	False	True	False	False	False	False	True	False
target_freq	0.85	0.90	0.85	0.80	0.80	0.80	0.85	0.85
generate_dummies	True	True	True	True	True	True	True	False

Table 4: *Best preprocessing parameters found for each model*

(a) *LDA*

solver	[svd]
priors	[None]
tol	[0.0001, 0.001, 0.01]

(b) *QDA*

priors	[None]
reg_param	[0.0, 0.1, 0.2, 0.3, 0.4, 0.5]
store_covariance	[True, False]
tol	[0.0001, 0.001, 0.01]

(c) *knn*

n_neighbors	[7, 9, 11, 13, 15, 17, 19]
weights	[uniform, distance]

(d) *logistic regression*

penalty	[l1, l2]
C	[0.1, 1, 10]
class_weight	[None, balanced]
intercept_scaling	[0.1, 1, 10]
max_iter	[1000]
solver	[saga, newton-cg]

(e) *SVM*

penalty	[l1, l2]
C	[0.1, 1, 10]
intercept_scaling	[0.1, 1, 10]
class_weight	[None, balanced]
max_iter	[1000]

(f) *random forest*

n_estimators	[50, 75, 100, 125]
criterion	[gini]
max_depth	[None, 25, 30, 35, 40]
max_features	[sqrt, log2]
class_weight	[None, balanced]

(g) *xgboost*

n_estimators	[65, 75]
max_depth	[None]
learning_rate	[0.2, 0.3, 0.4]
reg_lambda	[0.2, 0.3, 0.4]
reg_alpha	[0.2, 0.3, 0.4]

(h) *catboost*

iterations	[500, 750]
depth	[1, 2, 4, 6]
border_count	[32, 64, 96]
loss_function	[Logloss]
eval_metric	[F1, AUC]

Table 5: *Hyperparameters tested in cross-validation for each method. The yellow underlines refer to final chosen optimal hyperparameters*

Table 6: *Education Level and Count for Census-KDD*

Education Level	**Count**
Doctorate degree(PhD EdD)	1263
Prof school degree (MD DDS DVM LLB JD)	1793
Masters degree(MA MS MEng MEd MSW MBA)	6541
Bachelors degree(BA AB BS)	19865
Some college but no degree	27820
Associates degree-academic program	4363
Associates degree-occup/vocational	5358
High school graduate	48407
12th grade no diploma	2126
11th grade	6876
10th grade	7557
9th grade	6230
7th and 8th grade	8007
5th or 6th grade	3277
1st 2nd 3rd or 4th grade	1799
Less than 1st grade	819
Children	47422

9.2 Plots

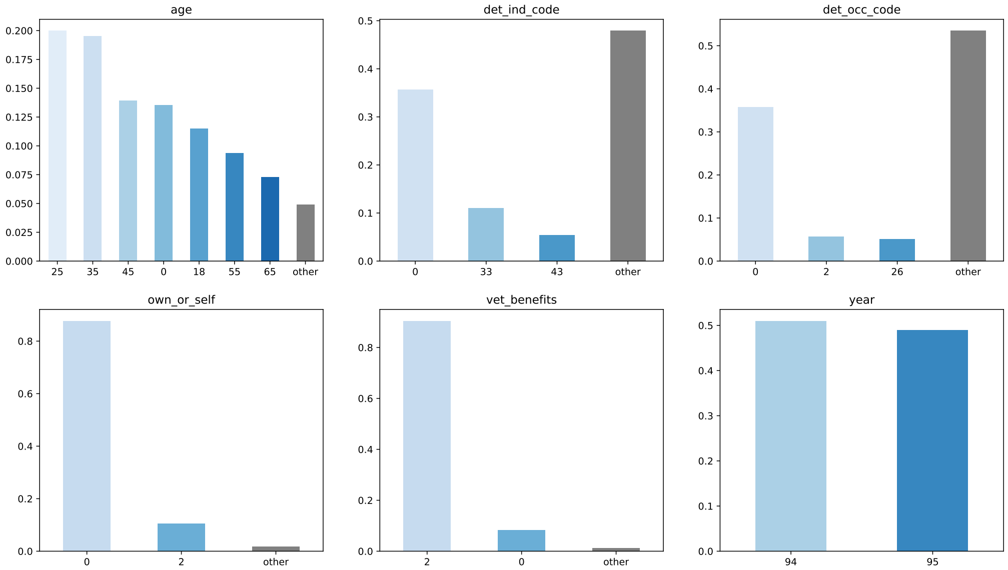


Fig. 10: *Resulted categorical data after conversion*

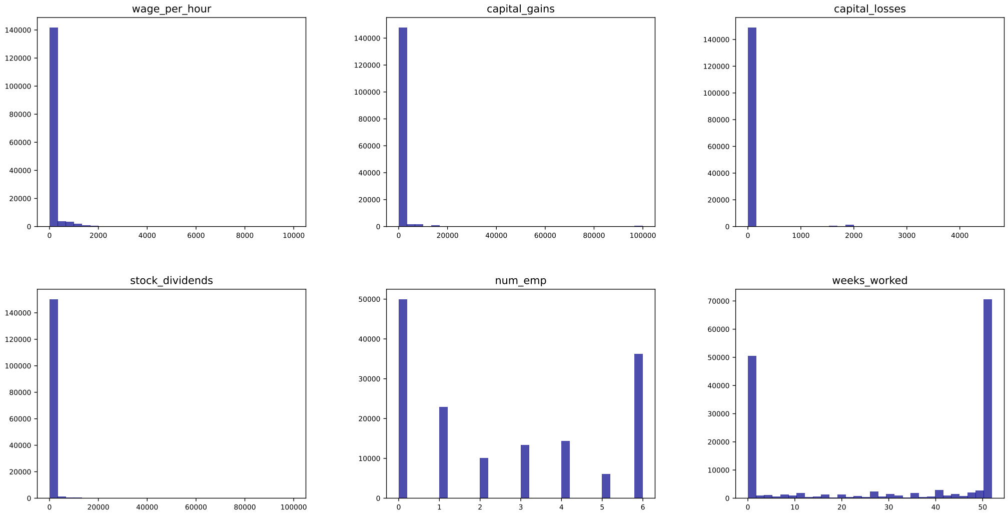


Fig. 11: *Histogram of the remaining numerical variables after categorical conversion*

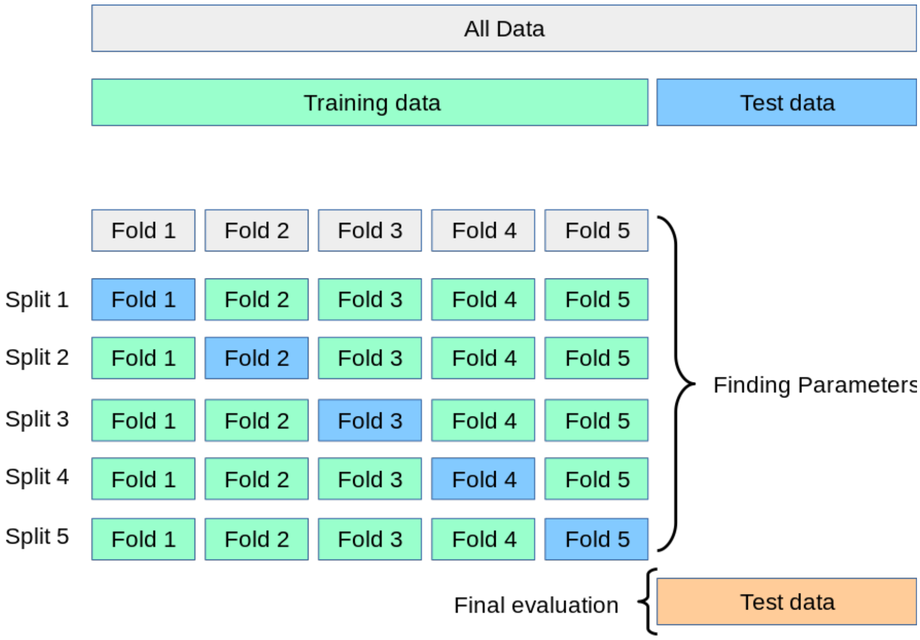


Fig. 12: *Cross-validation schema*

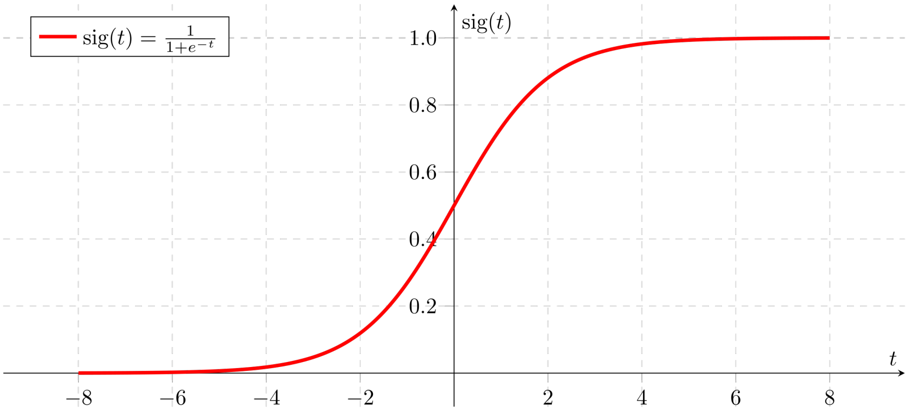
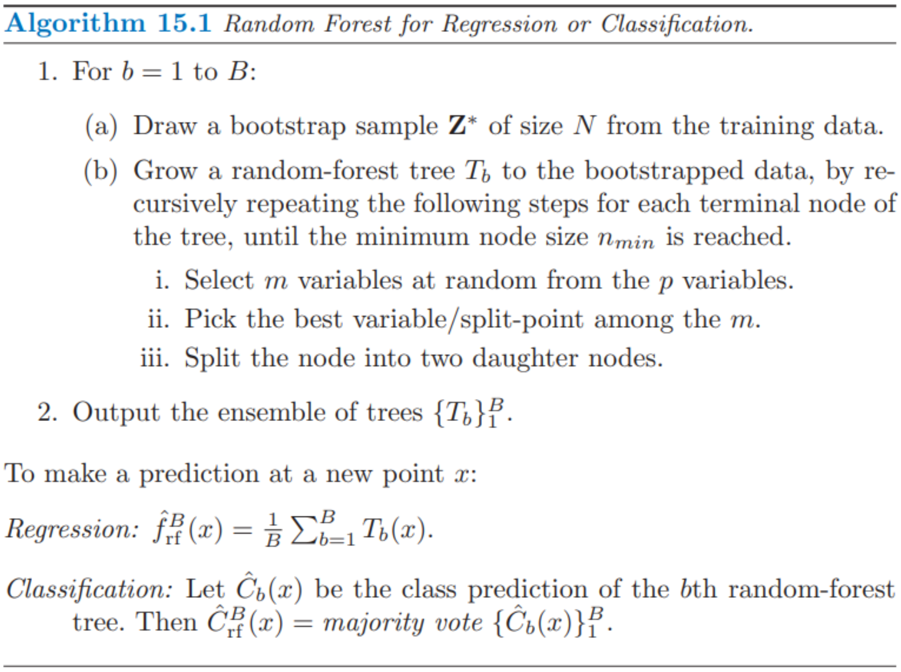


Fig. 13: *Sigmoid Function*



Algorithm 15.1 *Random Forest for Regression or Classification.*

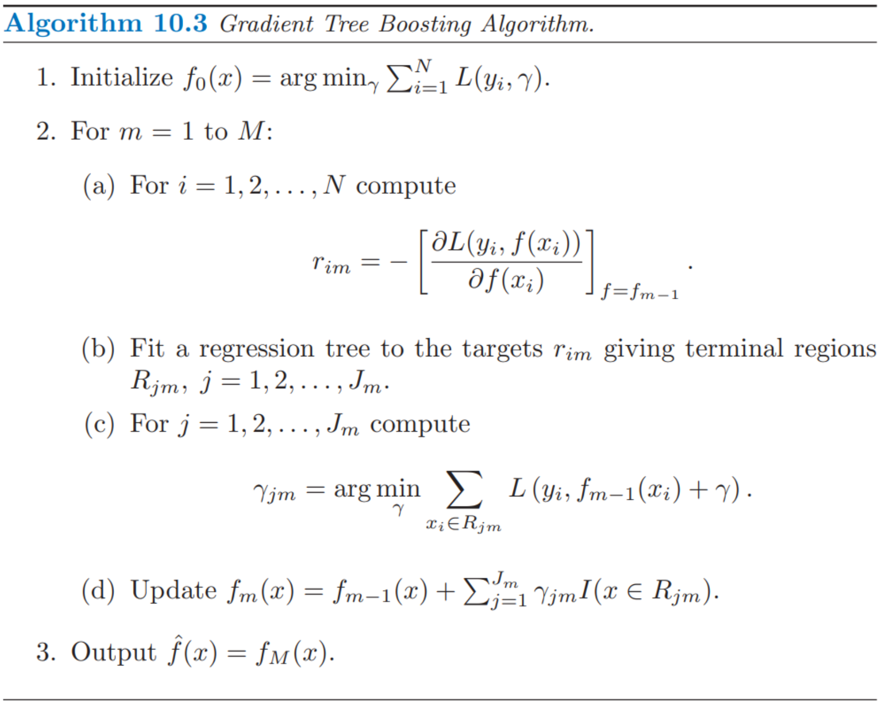
```
1. For $b = 1$ to $B$:
   (a) Draw a bootstrap sample $\mathbf{Z}^*$ of size $N$ from the training data.
   (b) Grow a random-forest tree $T_b$ to the bootstrapped data, by recursively repeating the following steps for each terminal node of the tree, until the minimum node size $n_{min}$ is reached.
       i. Select $m$ variables at random from the $p$ variables.
       ii. Pick the best variable/split-point among the $m$.
       iii. Split the node into two daughter nodes.
2. Output the ensemble of trees $\{T_b\}_1^B$.
```

To make a prediction at a new point x:

Regression: $\hat{f}_{\rm rf}^B(x) = \frac{1}{B}\sum_{b=1}^B T_b(x)$.

Classification: Let $\hat{C}_b(x)$ be the class prediction of the bth random-forest tree. Then $\hat{C}_{\rm rf}^B(x) = \textit{majority vote}\ \{\hat{C}_b(x)\}_1^B$.

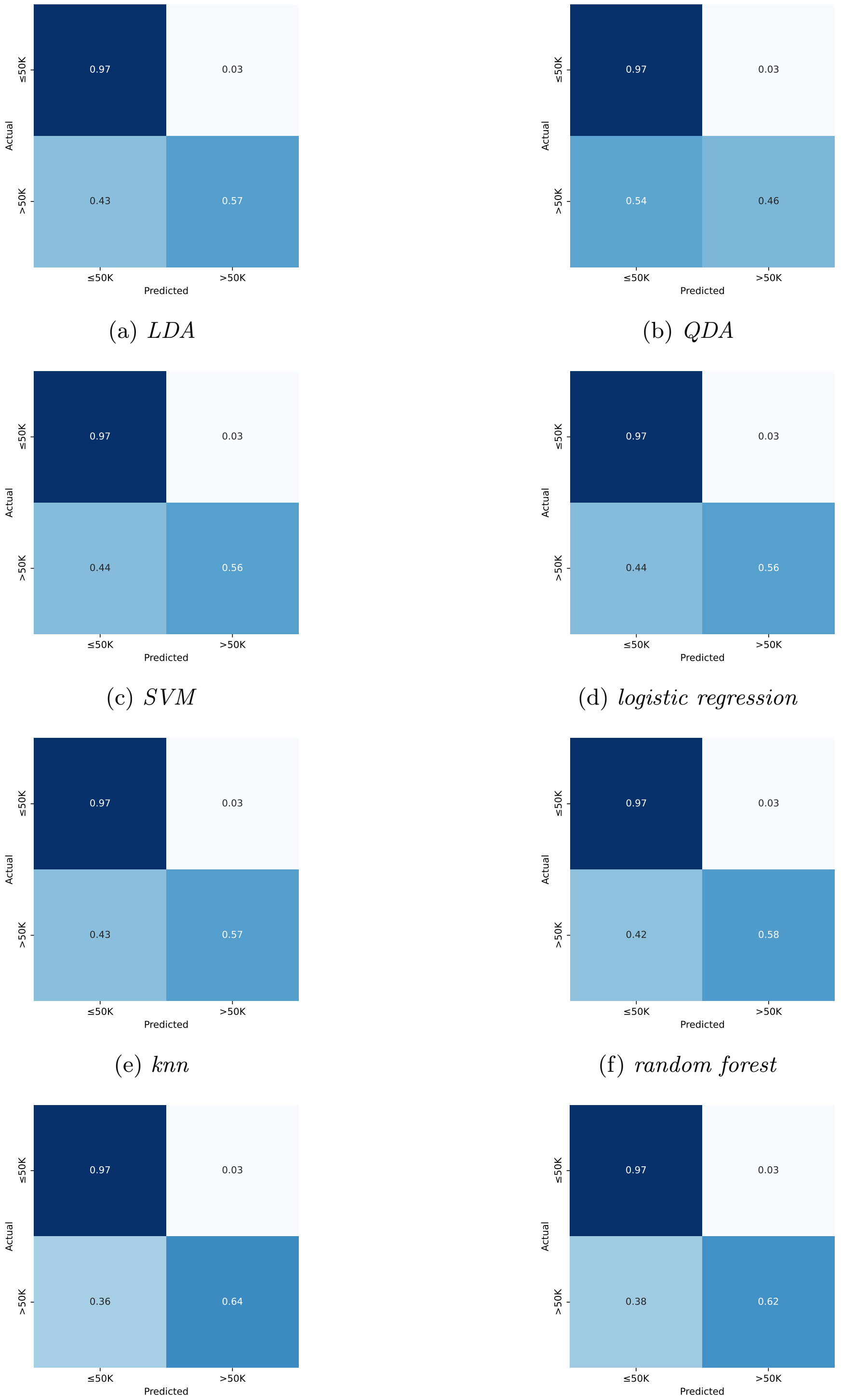
Fig. 14: *Step-by-step random forest algorithm*



Algorithm 10.3 *Gradient Tree Boosting Algorithm.*

```
1. Initialize $f_0(x) = \arg\min_\gamma \sum_{i=1}^N L(y_i, \gamma)$.
2. For $m = 1$ to $M$:
   (a) For $i = 1, 2, \ldots, N$ compute
       $$r_{im} = -\left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f=f_{m-1}}.$$
   (b) Fit a regression tree to the targets $r_{im}$ giving terminal regions $R_{jm},\ j = 1, 2, \ldots, J_m$.
   (c) For $j = 1, 2, \ldots, J_m$ compute
       $$\gamma_{jm} = \arg\min_\gamma \sum_{x_i \in R_{jm}} L\left(y_i, f_{m-1}(x_i) + \gamma\right).$$
   (d) Update $f_m(x) = f_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$.
3. Output $\hat{f}(x) = f_M(x)$.
```

Fig. 15: *Step-by-step gradient boosting algorithm*



(a) *LDA*

(b) *QDA*

(c) *SVM*

(d) *logistic regression*

(e) *knn*

(f) *random forest*

(g) *catboost*

(h) *xgboosting*

Fig. 16: *Confusion matrix plot for all the models tested. The results were obtained using 20-fold cross-validation on the training dataset.*

9.3 Supplementary Content

Cross validation

Cross-validation is a technique used in machine learning to assess the performance and generalization ability of predictive models. It involves dividing the dataset into multiple subsets or folds. The model is trained on a portion of the data and evaluated on the remaining fold. This process is repeated multiple times, with each fold serving as the validation set. The evaluation scores are then aggregated to estimate the model's overall performance. Cross-validation helps account for data variability, provides a more reliable performance assessment, and aids in detecting overfitting. It is a valuable tool for model selection, and can also be used to find the optimal parameters of the model.

Expected loss in Discriminant Methods

The expected loss of assigning x to the category $G = k'$

$$\mathbb{E}_G[L(G,k')] = \sum_{k\in G} L(k,k')Pr(G=k|x) = \sum_{k\neq k'} Pr(G=k|x) = 1 - Pr(G=k'|x)$$

The assigned category is the one that minimizes the expected loss

$$\hat{k} = \underset{k}{\operatorname{argmin}}\, 1 - Pr(G=k|x) = \underset{k}{\operatorname{argmax}}\, Pr(G=k|x)$$

Minkownski distance

$$d(x,y) = ||x-y||_p = \left(\sum_{j=1}^{d} |x_j - y_j|\right)^{\frac{1}{p}}$$

Gini Index

Gini index measures the impurity of the partitions made by random forest, a lower value implies higher purity. It is differentiable and hence more amenable to numerical optimization. It is given by the following formula:

$$\sum_{k=1}^{K} \hat{p}_k(1-\hat{p}_k)$$

Here, $\hat{p}_k$ is the proportion of samples of the class k that are in the region we are considering.

Dual formulation of the SVM

$$\begin{aligned} \max_{\lambda} \quad & \lambda^T\mathbf{e} - \frac{1}{2}\lambda^T Y A A^T Y \lambda \\ \text{s.t.} \quad & \lambda^T Y \mathbf{e} = 0 \\ & 0 \leq \lambda \leq \nu \end{aligned}$$

It can be observed that the problem can be rewritten by substituting $Q = YAA^TY$, where Q will have a size of $m \times m$. When a large number of data points are considered for optimization, the computational complexity can become a challenge due to the quadratic calculations and potentially disproportionate matrix dimensions. For instance, in our dataset if 100,000 data points are used for SVM optimization, the matrix Q would have $10^5 \times 10^5$ elements, rendering it computationally infeasible.

Gini-based

In each tree constructed in the forest, we can measure the improvement in split as the importance of the variables. For example, if a variable achieves a very low Gini index in a split, indicating high purity, it means that the variable effectively separates the data and explains the outcome. This measure accumulates throughout the independently constructed trees.

Random Permutation Based

This approach is based on out-of-bag (OOB) samples. When building a tree, the OOB samples are passed through the tree, and the accuracy is recorded. Then, the values of variable j are permuted for all samples, and the permuted samples are passed through the tree again. The decrease in accuracy resulting from the permutation is recorded for all trees in the forest, and the importance of variable j is measured. If the accuracy decreases significantly, it indicates that the variable is important in predicting whether the income is above or below 50k. Conversely, if the accuracy changes very little, it means that the variable is not highly important.

9.3.1 Model assumptions

In this subsection, we will discuss the assumptions of each model (if they have any) and assess whether our data meets or approximates those assumptions.

Discriminant Analysis

Discriminant analysis operates under the assumption that the class-conditional distributions $f_k(x)$ follow a multivariate normal distribution. However, this assumption does not hold true in our dataset. As observed during the preprocessing phase, there are numerous categorical variables, and the continuous/numerical variables do not adhere to a normal distribution due to severe data imbalance.

The covariance matrices play a crucial role in determining whether to utilize Linear Discriminant Analysis (LDA) or Quadratic Discriminant Analysis (QDA). In LDA, we assume that the variance matrix is the same for all categories, while in QDA, we make the assumption that each category has a distinct covariance matrix. Consequently, the discriminant function transforms into a quadratic function of x, as certain terms are no longer constant.

In our dataset, we calculated both covariance matrices for each group from the samples. We observed that the covariance matrices are not significantly equal.

Logistic regression

Logistic regression assumes minimal or no multicollinearity among the independent variables. In our dataset, there is no evidence of multicollinearity, at least between the numerical features. The maximum VIF is 1.89, indicating low multicollinearity. However, the vast majority of variables are categorical, and many of them are interdependent, leading to collinearity. For instance, variables like *education* and *age* are closely linked, along with many other examples.

Logistic regression generally requires a large sample size to make accurate predictions. Fortunately, our dataset contains a substantial number of samples, which is advantageous for training logistic regression models effectively.

It also assumes that observations are independent of each other. In our dataset, we have taken steps during preprocessing to ensure independence among the samples like eliminating repeated rows.