To predict whether or not an individual received a medium-level income, we choose to implement a Random Forest (RF) for Classification. The RF algorithm is an ensemble method that is relatively efficient and fast. Ensemble methods are powerful classification tools that combine individual decision trees, and the RF in particular exhibits low correlation between the individual trees (as each tree is based on a different set of features), which reduces the variance. We implement the RF using `sklearn.RandomForestClassifier`.

We split our data into an estimation and validation set, such that we can train our model and test its out-of-sample (OOS) accuracy. We choose to do so with K-fold cross-validation because it prevents selection bias and increases the generalization performance. We choose $K = 5$ because given the size of the dataset, a larger number of folds can be computationally expensive, since every addition in hyperparameter tuning results in an exponential increase of the computation time. Also, this may result in higher variance (albeit lower bias). Overall, considering the bias-variance trade-off, 5 or 10 folds are recommended (Hastie et al., 2009).

The original dataset contains 158 features, hence including all of them may lead to overfitting as the model would become overly flexible. To address this, we use subset selection by assessing feature importance based on the mean decrease in impurity (Breiman, 2001). We choose this method because it combines well with RFs compared to alternatives and is computationally efficient. To prevent data leakage, we fit a decision tree (with default hyperparameters) separately for each fold of the cross-validation process and rank the features based on their importance. We then apply a threshold of 0.010 for feature importance, meaning that only features with an importance above this value are included in the model. This approach allows us to focus on the features that genuinely contribute to model performance, offering more flexibility than simply imposing a fixed number of features. Additionally, we set a lower bound of 25 features during regularization, primarily for computational efficiency, as this significantly reduces the number of iterations required during cross-validation. This lower bound is also empirically justified—when testing the model without hyperparameter tuning, we observe the highest accuracy above 25 features.

We proceed with the hyperparameter tuning as follows: for each additional feature in the fold, we also tune the model's hyperparameters based on a grid with the most common hyperparameter choices. Specifically, we tune the `n_estimators`, `max_depth`, `min_samples_split` and `min_samples_leaf`. We choose to tune these hyperparameters because they greatly impact the resulting model and its prediction, and specifically deal with overfitting, which is a major concern. Due to computational constraints, we are unable to tune every hyperparameter. For those not tuned, we rely on educated decisions based on theoretical considerations. We choose cross-entropy as the `criterion` because it is more sensitive to probability estimates and, while more expensive than Gini, yields slightly better results (Hastie et al., 2009). We use the square root for `max_feautures` because it strikes a balance between reducing overfitting and maintaining computational efficiency, and during initial tuning *sqrt* and *log*2 gave the same result.

After tuning, we come to the conclusion, based on accuracy, of using 32 features, with the hyperparameters in Table 1, yielding an OOS accuracy of 81.5% (Figure 1). The prediction accuracies in Breiman (2009) vary between 75.4% and 95.8%, meaning the RF we developed performs rather well. We rank the feature importance of the entire training set based on the same method of mean decrease in impurity (Figure 2), and select the top 32 features. We fit these features of the entire training set with the selected hyperparameters, and predict Y in the test set.

Improvements to this method include assessing feature importance alongside hyperparameter tuning and incorporating additional hyperparameters, such as a cost complexity parameter, if computational limitations allow. We could also expand the grid search to test more values, consider feature correlations, and implement regularization instead of subset selection (Hastie et al., 2009).

Appendix

Table 1: Selected hyperparameters (bold) with ranges applied for the grid search, default values in grey

parameter	grid
`n_estimators`	{100,**300**}
`max_depth`	{10, **None**}
`min_samples_split`	{**2**, 10}
`min_samples_leaf`	{**1**, 5}
`criterion`	entropy
`random_state`	0
`max_features`	sqrt
`min_weight_fraction_leaf`	0
`max_leaf_nodes`	None
`min_impurity_decrease`	0
`bootstrap`	True
`oob_score`	False
`n_jobs`	None
`verbose`	0
`warm_start`	False
`class_weight`	None

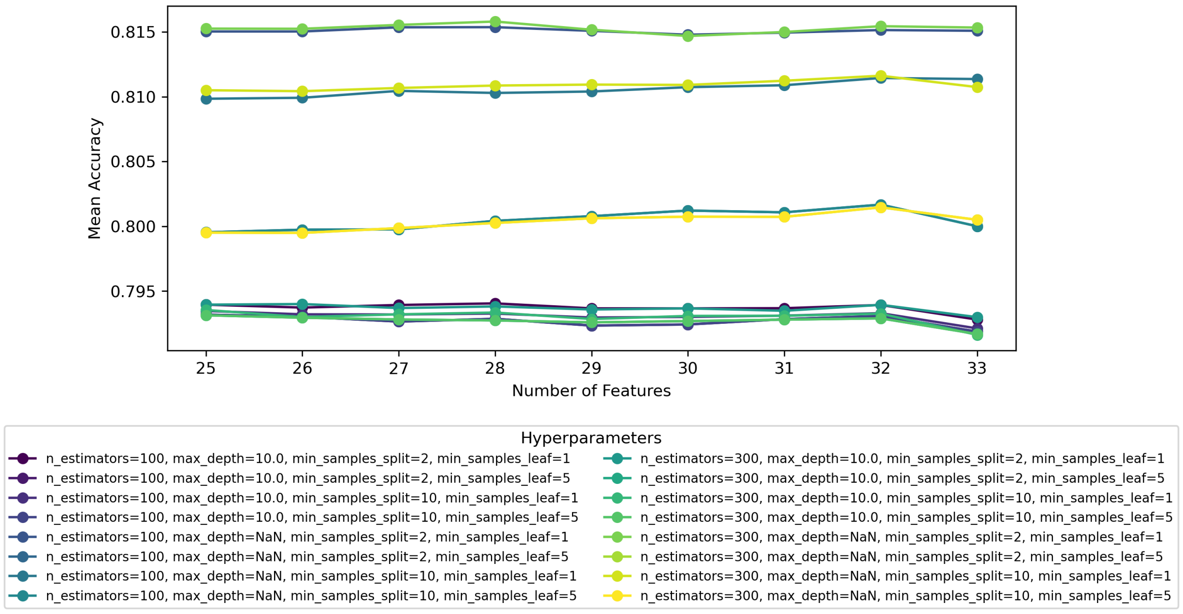

Figure 1: Mean accuracies over different hyperparameter combinations, averaged across 5 folds

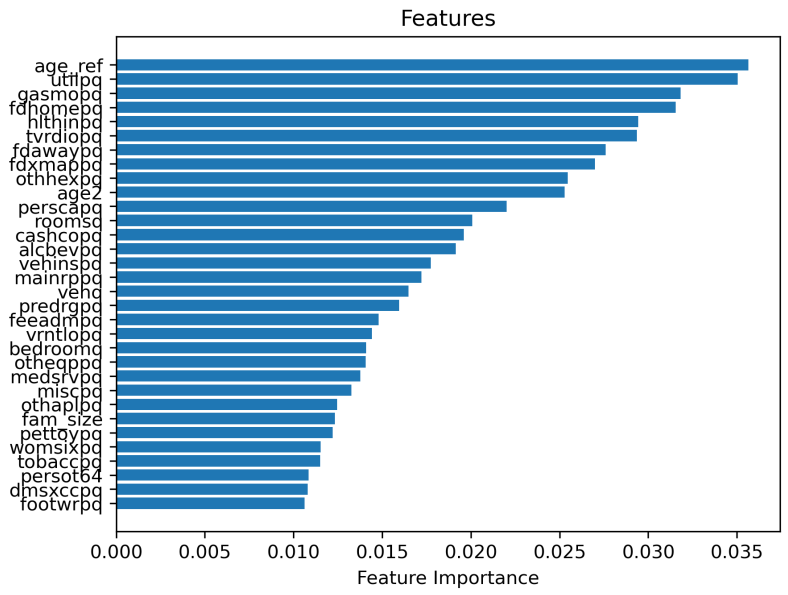

Figure 2: The 32 most important features subset used to train the RF model for predictions, based on mean decrease in impurity

References

Breiman, L. (2001). Random forests. 45.

Hastie, T., Tibshirani, R., Friedman, J. H., and Friedman, J. H. (2009). *The elements of statistical learning: data mining, inference, and prediction*, volume 2. Springer.