Random Forest And AdaBoost

Model Comparison

(Charlie) Xue Shun Hu
Gelderland, The Netherlands
Charlie.hu@ru.nl
Radboud University

Thomas Alberts
Utrecht, The Netherlands
Thomas.Alberts@ru.nl
Radboud University

ABSTRACT

In the domain of predictive modeling and machine learning, the task of income prediction has captured significant attention due to its implications in various fields, including finance, marketing, and social sciences. The ability to accurately estimate an individual's income can unlock insights into economic trends, consumer behavior, Credit Risk Assessment, Healthcare Resource Allocation, and societal dynamics. Therefor researchers and data scientists have delved into constructing models that can effectively predict income levels based on a diverse set of features.

In this study, we investigate if we can predict income levels based on a plethora of attributes. Utilizing different algorithms, our objective is to identify the algorithm that yields the highest accuracy score in predicting a person's income. Subsequently, we aim to perform a comparative analysis to discern the differences between the outcomes produced by these algorithms. We prepared the data by splitting it into 20% for testing and 80% for training. GridSearchCV was applied on the training set (80%) to determine the best parameters for each algorithm. Subsequently, these optimized parameters were tested on the remaining 20% for evaluation. We found that AdaBoost outperforms Random Forest in income prediction up to a max_depth of 8. Beyond that, Random Forest is slightly better, although the difference is not as pronounced as it is before a max_depth of 8. AdaBoost tree is smaller, thus enabling faster data processing. Furthermore, we found that AdaBoost performs better at predicting values less than or equal to 50k, while Random Forest excels at predicting values greater than 50k.

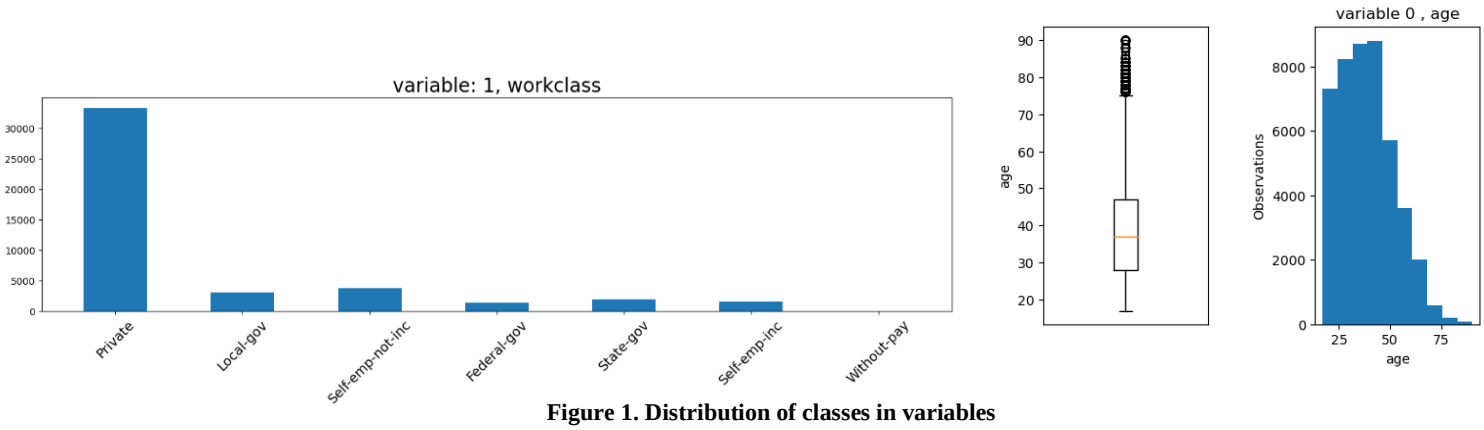


Figure 1. Distribution of classes in variables

Data specifics

The data used in this research is taken from the website *kaggle.com*, which obtained the data from the UCI repository. The original dataset was compiled by the US Census Bureau. The data was selected based on its availability and the number of elements in the dataset. This research was conducted to show our understanding of data visualisation and model comparison. Therefore, the significance of answering the research question should not be overstated, as the process and comparison is the main objective of this paper.

The dataset contains numerous variables. It contains both categorical, e.g. workclass, education, marital status; and numerical ones, e.g. age, fnlwgt, capital gain. Listed above are some distributions of the dataset according to some of these variables. The numerical variables are visualised using a boxplot and histogram, whereas the categorical ones are visualised using a bar chart.

Some notable insights are that the dataset contains a considerable proportion of the class White in the variable race. The same holds for the class private in the variable workclass.

After the generation of the graphs, some categorical variables contained the class "?". To ensure classifiers do not make predictions based on this unknown class, approximately 2500 rows containing these datapoints were removed. The histograms, boxplots and bar charts are generated after the removal of these datapoints.

1. Research method

1.1 Objective

The objective of the classifiers is to give a classification of a person's income based on variables related to them. Through this, conclusions can be based on the most important decisions made by the classifiers based on these variables.

Thus the classifiers should have the objective to predict whether a person earns ">50k" or "<=50k", based on the other variables.

">50k" shall be transformed to 1, and "<=50k" shall be 0. This is needed for the binary method of classification.

To reduce the variability of the two classifiers when comparing them, it is imperative to only adjust hyperparameters that both have in common. Adjusting specific hyperparameters that one classifier has, but the other doesn't creates unbalance in the comparison. For this reason, all other hyperparameters are left as default according to the scikit library.

1.2 Classifiers

The classifiers used are **Random Forest** and **AdaBoost**. Both are implemented in python via the sklearn.ensemble module in the scikit-learn library in python.

Random Forest is a classifier based on multiple decision trees. It makes multiple decision trees, which are trained using a bootstraps of the train data and in which each only a subset of the variables are used. After the trees are build, a prediction on new a person is based on the majority vote of these trees. If the majority of the trees classify the person as ">=50k", then random forest will predict as such.

AdaBoost operates similarly to **Random Forest** in the sense that it uses a majority voting to decide on the classification. However, **AdaBoost** does not make complete trees. It creates **stumps**, which take only one variable into account. During the first iteration, it looks which variable creates the best split based on the metric used to evaluate a split. After the first iteration, a bootstrap is done on the train dataset to determine the next split. The bootstrap however now has weights based on whether or not the datapoint was incorrectly classified in the previous split. If a datapoint was incorrectly identified, then the weight is increased, increasing its chance to appear multiple times in the bootstrap. If it was correctly identified, the weight is decreased. By following this pattern, **AdaBoost** tries to adaptively increase its accuracy, by looking at what it incorrectly identified previously and trying to identify those datapoints. Then, when multiple of these **stumps** **AdaBoost** is finished.

When classifying a new datapoint, the algorithm examines how the stumps classify the datapoint. Not all stumps have an equal amount of say in this, not all **stumps** contribute equally.. The ones that have a low **error rate,** the amount of datapoints it misclassified, will have a higher impact, than those with a high **error rate**.

1.3 One-hot encoding

Before **Random Forest** can be applied on the variables to classify a datapoint, the categorical variables need to be transformed in to numerical ones. This is done using **one-hot encoding**.

One-hot encoding is essentially transforming a categorical variable into multiple categorical variables based on the number of classes that variable has. The new categorical variables will then have the name: "categorical variable_class name". These new categorical variables will then have a 1 if in the original variable the class name was the class of that datapoint, and 0 otherwise. No information is lost, however the amount of variables is increased significantly.

1.4 Train test split

Before training the models, the dataset is first split into train and test. Eighty percent of the dataset is used for training and the remaining twenty percent is used for testing. The split is stratified, which means that the train and test dataset both contain the same proportion of datapoints which have as classification "<50k" and ">=50k".

1.5.1 Hyperparameter optimization

Random Forest and **AdaBoost** only have 2 hyperparameters in common. These are **max_depth** and **n_estimators**. For **max_depth** the values to inspect will be: "1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 15, 20, 25" and for **n_estimators**: "10, 50, 100, 150, 200". These are frequent values which are used for these hyperparameters.

The combination of optimal hyperparameter values is based on the accuracy of the classifiers trained using that combination.

1.5.2 Train accuracy – 10-fold Cross-Validation

The train set will be used to train and validate the accuracy of the model using the hyperparameter combination. This will be enabled using a stratified 10-fold Cross-Validation on the train set. This operation enables us to use the entire train set to both train and validate. The process operates as follows: the dataset is first split in 10 parts, each of equal size and stratified. One part is set aside for validation, and the rest is used for training the classifier. This process repeats itself 10 times, after which all the parts have once been used as a validation set. The mean accuracy across all these validations is then used to represent the accuracy of the model using that combination of hyperparameters.

1.5.3 GridSearchCV

GridSearchCV is a sklearn.ensemble module from the scikit library. It enables searching for optimal hyperparameters by doing stratified 10-fold cross validation on all combination of hyperparameters. An example of how GridSearchCV operates can be seen in figure 2.

For max depth 3, takes an n_estimator amount, and does 10-fold cross validation to get the mean_test score.

From this, it can be seen that randomForest, with a max_depth of 3 has the highest mean_test_accuracy at the lowest amount of n_estimators. And AdaBoost at the value n_estimators equal to

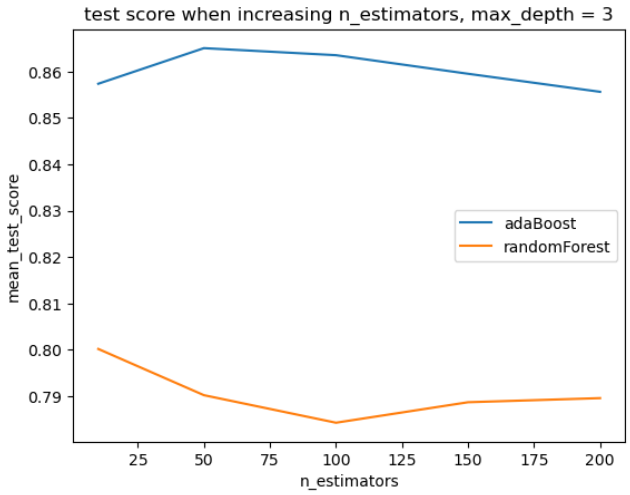


Figure 2. Grid search example accuracy, single combination. max_depth = 3, n_estimators varies

50. These best values are then compared against the accuracy of the best of other max_depth combinations.

The one that has the best mean_test_score will be the hyperparameters returned.

1.6 Mode evaluation

After the optimal hyperparameters are obtained, the trained models are then tested using the test set which has been set apart at the beginning. The evaluation of the model is completely based on the results of the prediction on the test set.

From this, the corresponding ROC-curve and AUC of each model are calculated.

The models are then further compared by looking at which datapoints one correctly identified and the other incorrectly identified.

Using these results, a two-sided sign test is done to evaluate whether or not one of these models performs better than the other.

2. Results

2.1 Optimal Hyperparameters: GridSearchCV

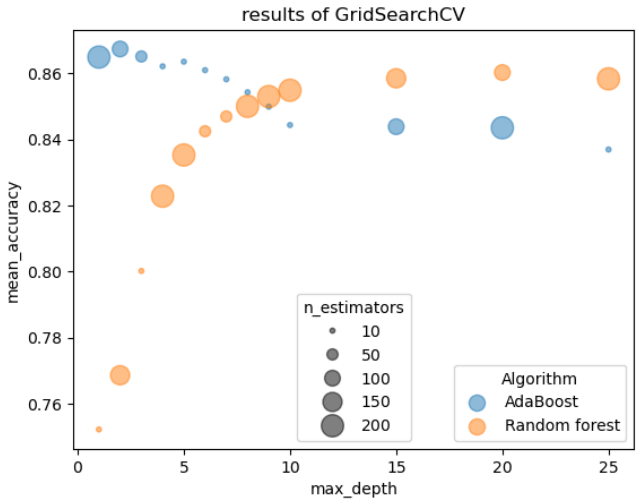


Figure 3: Overview of the highest accuracy scores obtained from each iteration of GridSearchCV on the train set. For each iteration of every classifier, the max_depth value was evaluated against each n_estimator. The pair that resulted in the highest accuracy score is depicted in the illustration. max_depth's used are 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 15, 20, 25.

In our research, we utilized GridSearchCV to optimize the hyperparameters of our classifiers. We evaluated the max_depth and n_estimators parameters, as these were the only common parameters across the classifiers, ensuring a fair comparison. The primary objective was to pinpoint the parameter combination that produced the highest accuracy score. This would not only provide insights into the optimal parameters for future use but also highlight the differences between the AdaBoost and Random Forest classifiers. **Our findings reveal that the combination yielding the highest accuracy score of 0.8673 is AdaBoost, specifically when max_depth is set to 2 and n_estimators is set to 100. In contrast, the highest result for Random Forest, 0.8602, is obtained when max_depth is 20 and n_estimators is 50**. It is observed that AdaBoost performs optimally with a lower max_depth, and its performance deteriorates with an increase in max_depth. On the other hand, Random Forest tends to favor a higher max_depth. However, its performance plateaus when max_depth becomes excessively high. Interestingly, after max_depth 8, Random Forest begins to outperform AdaBoost's best score per iteration. This comparison provides valuable insights into the behavior of these two algorithms under different parameter settings.

AdaBoost = [0.8673, 0.8651, 0.8649, 0.8635, 0.8621, 0.8609, 0.8582, 0.8542, 0.8499, 0.8443, 0.8438, 0.8435, 0.8369]

Random Forest = [0.8602, 0.8585, 0.8583, 0.8549, 0.8529, 0.8500, 0.8469, 0.8424, 0.8353, 0.8228, 0.8001, 0.7686, 0.7522]

The average accuracy for AdaBoost is calculated as the sum of AdaBoost scores divided by the length of the list, which equals 0.8550. Similarly, the average accuracy for Random Forest is 0.8310.
Upon comparing all the accuracy results, it is observed that AdaBoost outperforms Random Forest in this scenario, although the averages are fairly close. This observation would differ if we considered the results up to a max_depth of 8. In that case, AdaBoost would be favored and the difference in averages would be greater. However, if we consider the results after max_depth of 8, the average will favor Random Forest, albeit with a smaller difference compared to before a max_depth of 8.

One caveat to the use of GridSearchCV, was the max_depth could not be directly used in the parameter search. This is because not directly insertable into AdaBoost. Hence we opted for a for loop on the different values of max_depth, and inside it do a GridSearchCV on the best n_estimator value with that max_depth. Each result of the GridSearchCV, would then be the optimal n_estimator, given a max_depth. From this we took the parameters with the highest mean_accuracy.

2.2 Testing the results on 'real data'

With the best-performing classifier parameters in mind, we validated our results by testing on the 'real data'—the 20% of the dataset that we reserved for testing. We accomplished this by using each classifier to predict outcomes on the 20% data.
For AdaBoost, the highest accuracy achieved on the training data was 0.8673. When this model was used to predict outcomes on the 20% test data, the accuracy was 0.8647. For Random Forest, the highest accuracy achieved on the training data was 0.8602. When this model was used to predict outcomes on the 20% test data, the accuracy was 0.8590. Based on these results we created the project reports depicted below in figure 4.

Figure 4: AdaBoost and Random Forest report, for result comparison.

AdaBoost report:

	precision	recall	f1-score	support
0	0.89	0.93	0.91	6803
1	0.76	0.67	0.71	2242
accuracy			0.86	9045
macro avg	0.83	0.80	0.81	9045
weighted avg	0.86	0.86	0.86	9045

RandomForest report:

	precision	recall	f1-score	support
0	0.88	0.95	0.91	6803
1	0.78	0.60	0.68	2242
accuracy			0.86	9045
macro avg	0.83	0.77	0.79	9045
weighted avg	0.85	0.86	0.85	9045

We further analyzed these results by comparing them in the form of plotting the ROC curves, Confusion matrixes. These are depicted below, looking at the ROC curve in Figure 5, AdaBoost slightly outperforms Random Forest, and a similar trend is observed when analyzing the confusion matrixes in Figure 6 and 7. Furthermore, AdaBoost demonstrates superior performance in predicting class ">50k", whereas Random Forest excels in predicting class "<=50k".

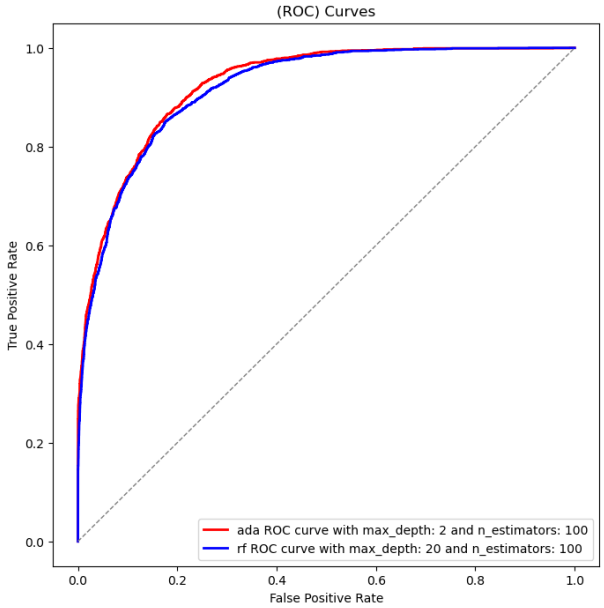


Figure 5: Comparison of AdaBoost ROC curve in red and Random Forest in blue.

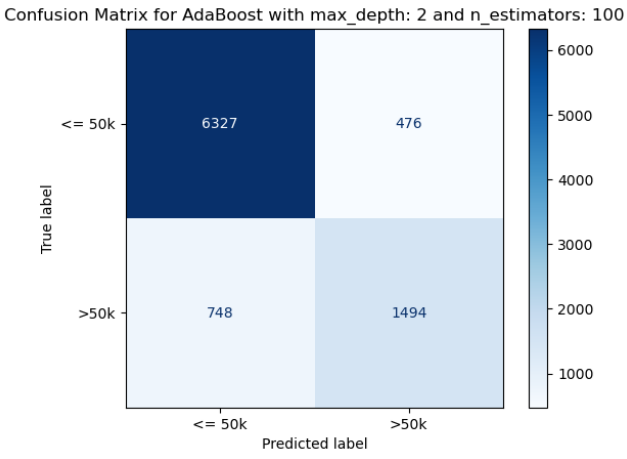


Figure 6: The confusion matrix for AdaBoost, which illustrates the results after predicting on the test data.

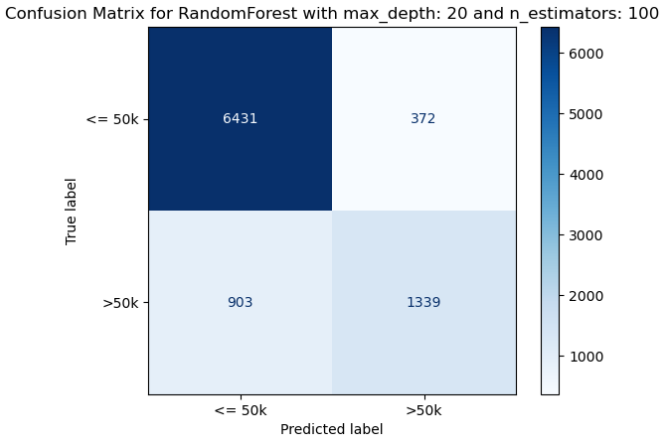


Figure 7: The confusion matrix for Random Forest, which illustrates the results after predicting on the test data.

Subsequently, we constructed Decision trees, fully aware of the fact that they would vary in size. By examining the nodes of Figure 8, we get insights into the initial steps of each classifier. The initial step of AdaBoost is to determine whether the individual is married. Then it proceeds to node #1 and checks if the capital gain is lower than 7055.50. If not, it advances to node #4 and examines the educational_num, which reflects the level of education. After that, AdaBoost has enough information to determine the class. For Random Forest, as depicted in Figure 9, we note that not all nodes are illustrated in this overview. If they were, it would render the figure unreadable. Furthermore, we believe that the initial steps are of greater importance because more data passes through them. In node #0, it checks to see if the individual works less than 41.5 hours per week. If so, it verifies if the individual's occupation is farming or fishing. If not, it checks if the relationship status is "other-relative".

Figure 8: AdaBoost Decision tree

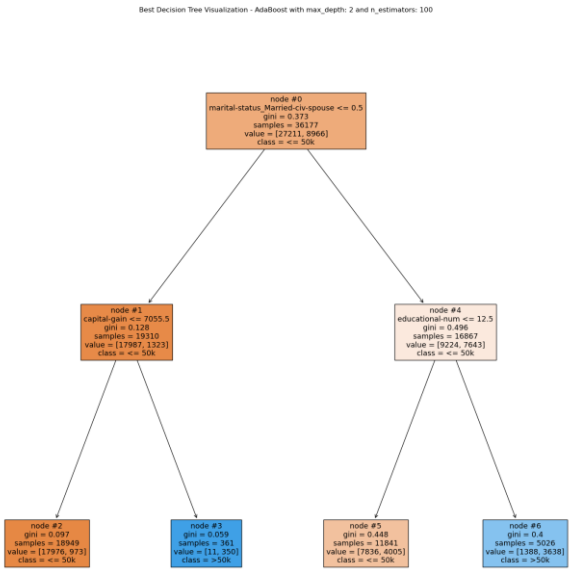
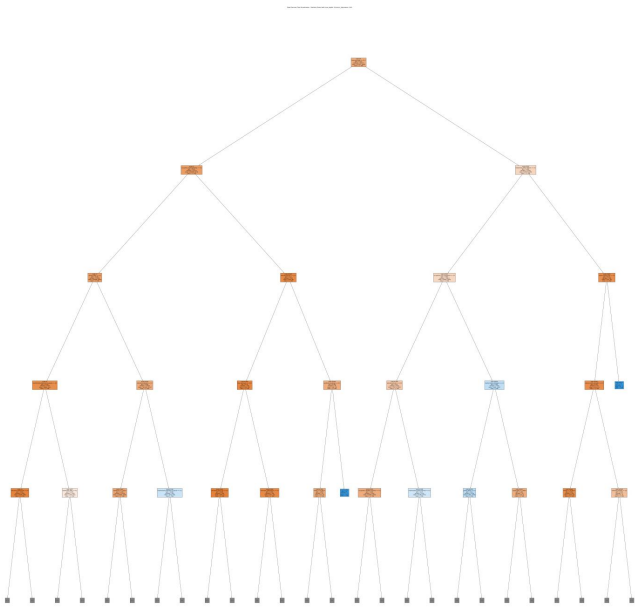


Figure 9: Random Forest Decision tree



Lastly, we compared the models directly in figure 10. The main purpose is to look in which test cases, one model deviated from the other with respect to the classification of a datapoint. These are the points of interest. In figure 10, we see that AdaBoost is correct 301 times, when Random Forest is wrong. And that Random Forest is correct 250 times, when Random Forest is wrong. To see if there is a significant difference between these two models, we do a two-sided sign test based on these values.

Figure 10: Comparison of AdaBoost and Random Forest results.

		randomForest	
		correct	wrong
adaBoost	correct	7520	301
	wrong	250	974

2.3 Two sided sign test

We expect the chance of either to be wrong or correct to be 50%. The null hypotheses would then be “There is no significant difference between the use of AdaBoost or Random Forest to classify this specific data.”. The significance level we set to the standard 0.05.

Calculating the p-value related to these findings results in p = 0.0330. This means that we reject the null hypothesis. There is a statistical significant difference between the observed results. This means there is statistical significant difference in the use between AdaBoost and Random Forest to classify this data.

Conclusion

AdaBoost appears to be superior based on all comparisons and tests. This is because it requires fewer decisions to make a choice, resulting in faster results and processing.

However, it should be noted that both AdaBoost and Random Forest have many more hyperparameters that can be tuned. Thus it is not definite that either one is better than the other overall with respect to the classification of this data.

Future Additions or Missed Opportunities

The dataset comprises 75% of rows with values less than or equal to 50k and 25% with values greater than 50k. A potential extension of this study could involve comparing the classifiers using a dataset where this split is 50%/50%, to determine if any significant differences emerge. This consideration is based on the observation that AdaBoost performs better at predicting values less than or equal to 50k, while Random Forest excels at predicting values greater than 50k.

Furthermore, tuning the other hyperparameters, that either are not common across both, will truly show which one is a better fit for this dataset.

Due to time constraints, we were not able to explore these details further.

We also were not able to give a concise answer to the research question.

Appendix

Dataset: https://www.kaggle.com/datasets/wenruliu/adult-income-dataset

Code clarification

In [11]:
```
# Gridsearch to find the optimal hyperparameters for each classifier
# rf classifier also included since rfclassifier can use in grid search but ada cant
# store the results of each Gridsearch, into a list, because of the max_depth not being gridsearchable.
# Standard 10-fold stratified cross validation in gridSearchCV
# gridSearchCV does 10-fold stratified cross, so there is no need to do testing again afterwards.


adaBestParamList = []
rfBestParamList = []

# Function to search for the best parameters via GridSearchCV using 10-fold cross validation, comparing result with best result
# max_depth is via loop because, Adaboost handles parameters diffrent from RandomForest
for i in hyper_params["max_depth"]:
# AdaBoost
    base_classifier_ada = DecisionTreeClassifier(max_depth=i, random_state=random_state)
    ada_classifier = AdaBoostClassifier(base_classifier_ada, random_state=random_state)

    grid_search_ada = GridSearchCV(ada_classifier, {'n_estimators': hyper_params['n_estimators']}, cv=10, n_jobs= -1)
    grid_search_ada.fit(X_train, y_train)
    adaIndex = [i for i, rank in enumerate(grid_search_ada.cv_results_['rank_test_score']) if rank == 1]
    adaBestRow = {key: val[adaIndex[0]] for key, val in grid_search_ada.cv_results_.items()}
    adaBestRow["max_depth"] = i
    adaBestParamList.append(adaBestRow)
    pd.DataFrame(grid_search_ada.cv_results_)

    best_ada_classifier = grid_search_ada.best_estimator_
    best_score_ada = grid_search_ada.best_score_
    accuracy_ada_list.append(best_score_ada)
    # Update overall best classifier
    update_best_results("ada", best_ada_classifier, best_score_ada, X_train, y_train)

    rf_classifier = RandomForestClassifier(max_depth=i, random_state=random_state)
    grid_search_rf = GridSearchCV(rf_classifier, {'n_estimators': hyper_params['n_estimators']}, cv=10, n_jobs= -1)
    grid_search_rf.fit(X_train, y_train)
    rfIndex = [i for i, rank in enumerate(grid_search_rf.cv_results_['rank_test_score']) if rank == 1]
    rfBestRow = {key: val[rfIndex[0]] for key, val in grid_search_rf.cv_results_.items()}
    rfBestRow["max_depth"] = i
    rfBestParamList.append(rfBestRow)

    best_rf_classifier = grid_search_rf.best_estimator_
    best_score_rf = grid_search_rf.best_score_
    accuracy_rf_list.append(best_score_rf)

    # Update overall best classifier
    update_best_results("rf", best_rf_classifier, best_score_rf, X_train, y_train)

    # Print example of GridSearchCV of how evaluates the best hyperparameters, based on input.
    if i == hyper_params["max_depth"][2] :
        meanTestScoresAda = grid_search_ada.cv_results_["mean_test_score"]
        parametersAda = grid_search_ada.cv_results_["param_n_estimators"]
        meanTestScoresRf = grid_search_rf.cv_results_["mean_test_score"]
        parametersRf = grid_search_rf.cv_results_["param_n_estimators"]
        fig = plt.figure()
        plt.plot(parametersAda, meanTestScoresAda, label = "adaBoost")
        plt.plot(parametersRf, meanTestScoresRf, label = "randomForest")
        plt.legend()
        plt.xlabel("n_estimators")
```

As stated in the paper, grid search could not take max_depth as a variable for AdaBoost. Hence we put this outside as a for loop. For each value of the max_depth, we do GridSearchCV to find the optimal n_estimator number. To give an example of how gridSearchCV works, we included the if statement at the bottom, to show a single example of what grid search computed for the third value of the max_depth we searched through.

Data overview text

age range: 17 to 90

percentage >50k: 24.78%, <=50k: 75.22%

Total >50k: 11208, <=50k: 34014

percentage White: 86.03%, Black: 9.35%, Asian-Pac-Islander: 2.88%, Amer-Indian-Eskimo: 0.96%, Other: 0.78%

Total White: 38903, Black: 4228, Asian-Pac-Islander: 1303, Amer-Indian-Eskimo: 435, Other: 353

percentage Male: 67.50%, Female: 32.50%

Total Male: 30527, Female: 14695

Highest education Level	>50K Percentage	<=50K Percentage
10th	6.704824	93.295176
11th	5.497221	94.502779
12th	7.452340	92.547660
1st-4th	3.603604	96.396396
5th-6th	4.899777	95.100223
7th-8th	6.682868	93.317132
9th	5.621302	94.378698
Assoc-acdm	26.410086	73.589914
Assoc-voc	25.727412	74.272588
Bachelors	41.981506	58.018494
Doctorate	73.345588	26.654412
HS-grad	16.343097	83.656903
Masters	55.409706	44.590294
Preschool	1.388889	98.611111
Prof-school	75.414013	24.585987
Some-college	20.103041	79.896959

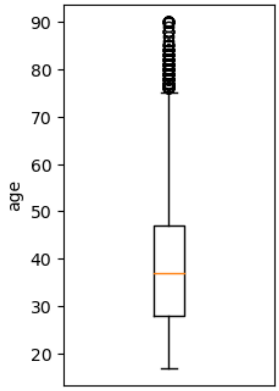
Highest education Level	Male Percentage	Female Percentage
10th	68.029436	31.970564
11th	65.657813	34.342187
12th	68.977470	31.022530
1st-4th	74.774775	25.225225
5th-6th	75.723831	24.276169
7th-8th	76.063183	23.936817
9th	72.337278	27.662722
Assoc-acdm	61.181155	38.818845
Assoc-voc	65.543645	34.456355
Bachelors	69.220608	30.779392
Doctorate	80.147059	19.852941
HS-grad	68.470540	31.529460
Masters	68.058870	31.941130
Preschool	72.222222	27.777778
Prof-school	84.203822	15.796178
Some-college	62.481059	37.518941

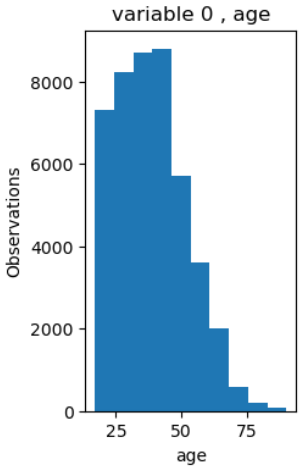
Race	>50K Percentage	<=50K Percentage
Asian-Pac-Islander	28.319263	71.680737
White	26.237051	73.762949
Other	12.747875	87.252125
Black	12.630085	87.369915
Amer-Indian-Eskimo	12.183908	87.816092

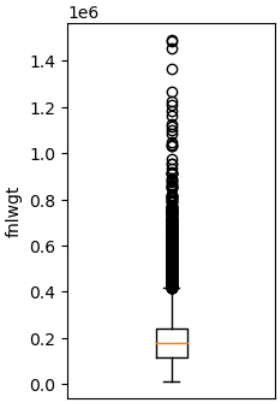
Race	Male Percentage	Female Percentage
Amer-Indian-Eskimo	61.839080	38.160920
Asian-Pac-Islander	66.538757	33.461243
Black	50.709555	49.290445
Other	64.305949	35.694051
White	69.454798	30.545202

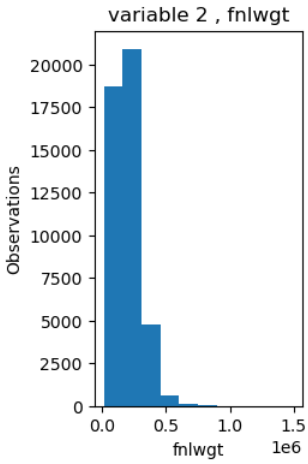
Gender	>50K Percentage	<=50K Percentage
Male	31.247748	68.752252
Female	11.357605	88.642395

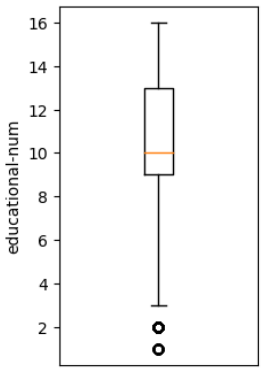
Graphs

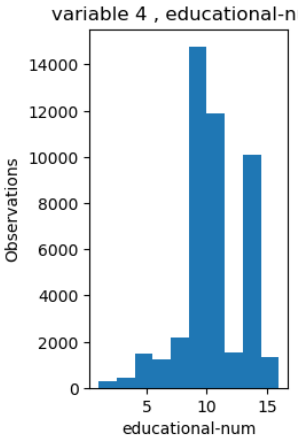

age

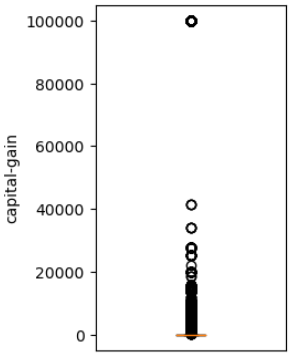

variable 0 , age
Observations
age

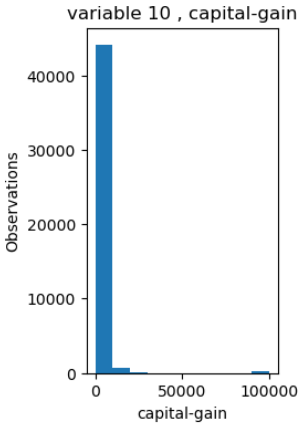

1e6
fnlwgt

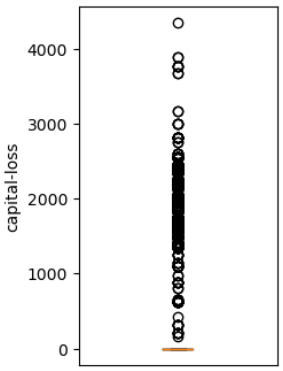

variable 2 , fnlwgt
Observations
fnlwgt
1e6

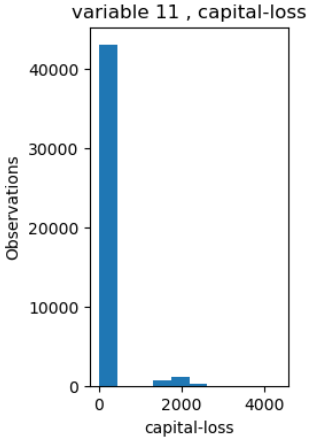

educational-num

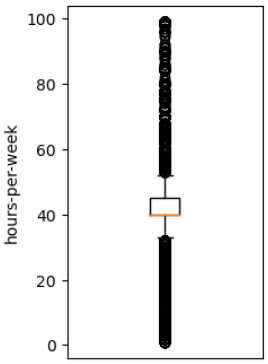

variable 4 , educational-num
Observations
educational-num

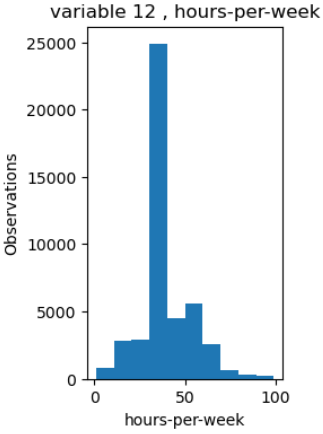

capital-gain

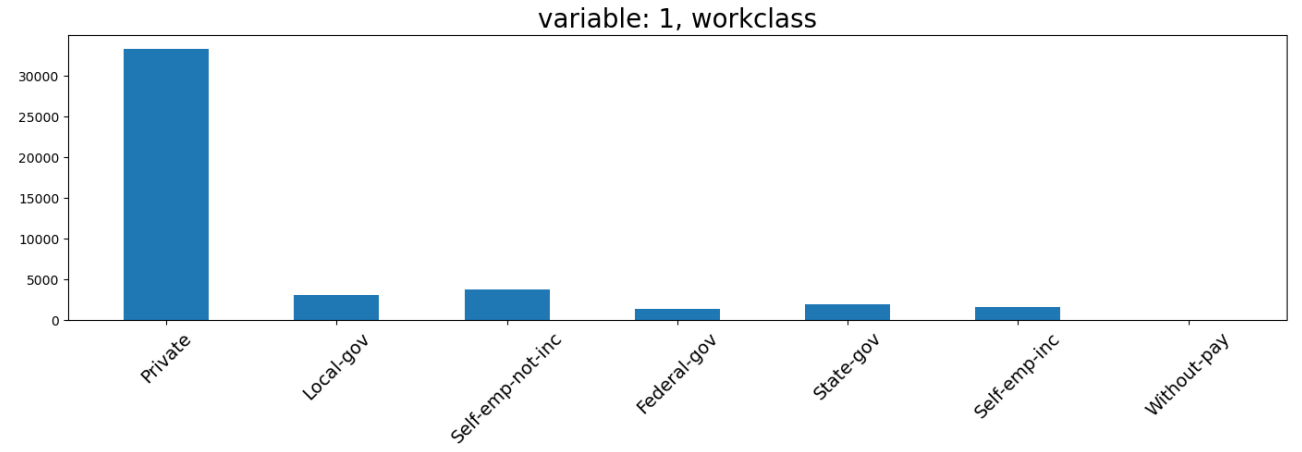

variable 10 , capital-gain
Observations
capital-gain

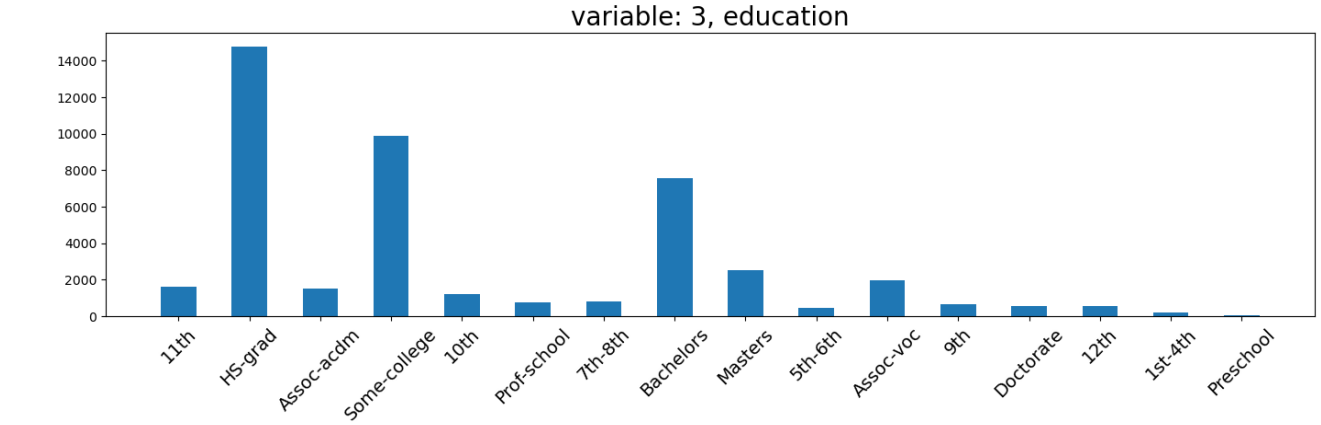

capital-loss

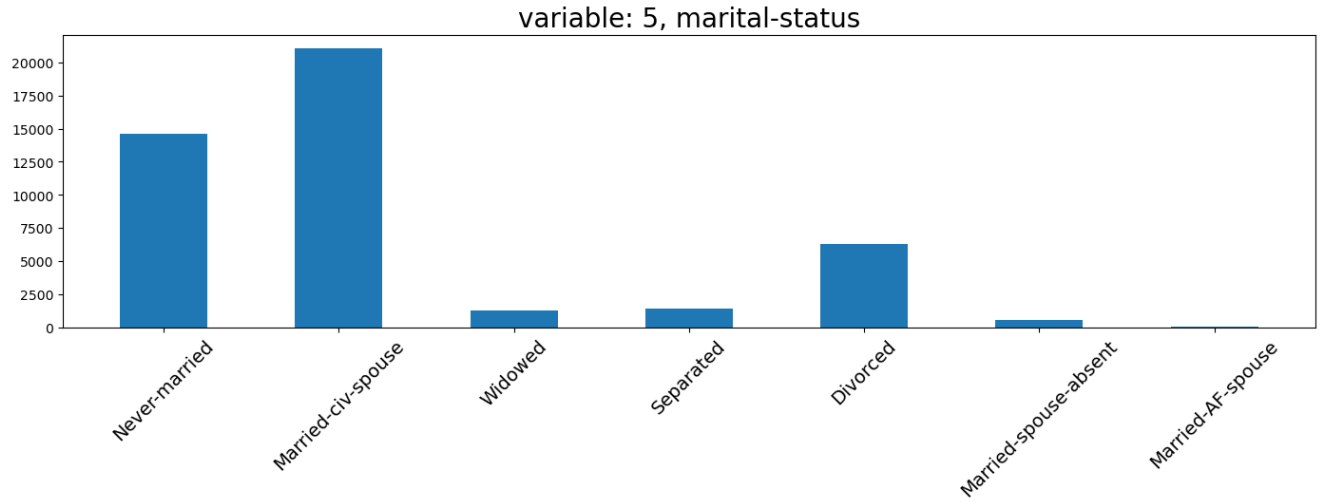

variable 11 , capital-loss
Observations
capital-loss

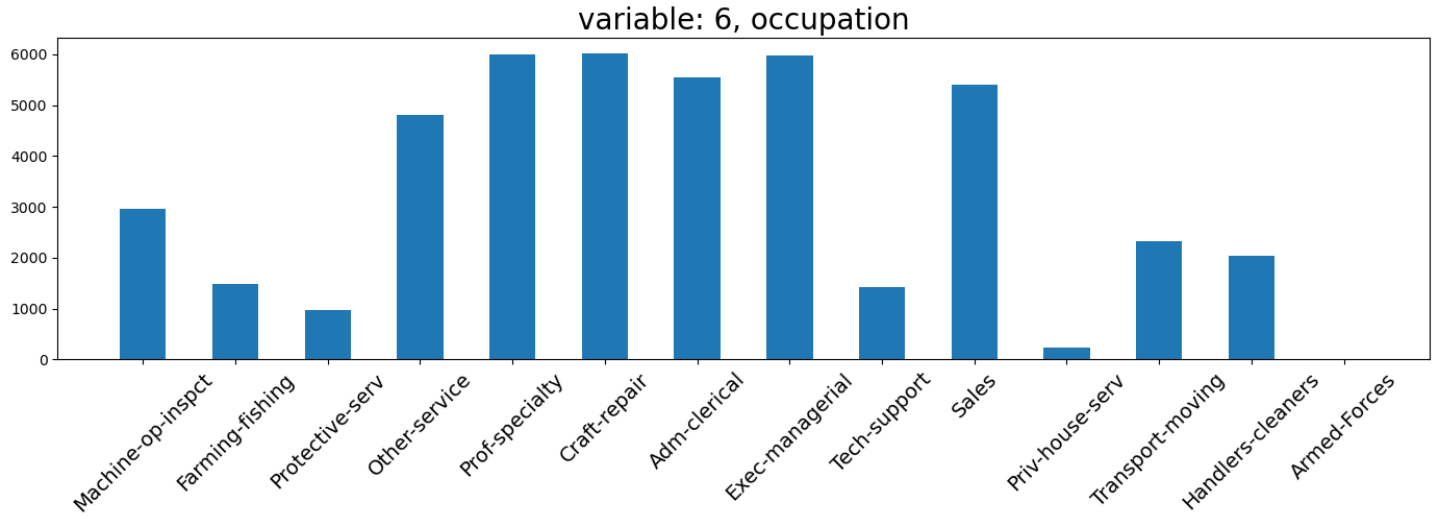

hours-per-week

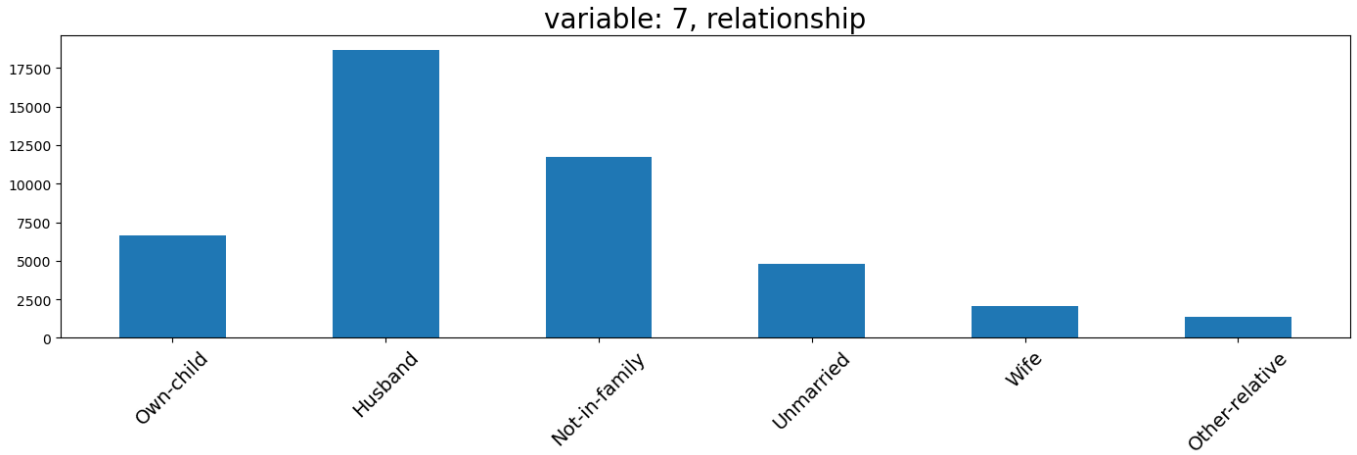

variable 12 , hours-per-week
Observations
hours-per-week

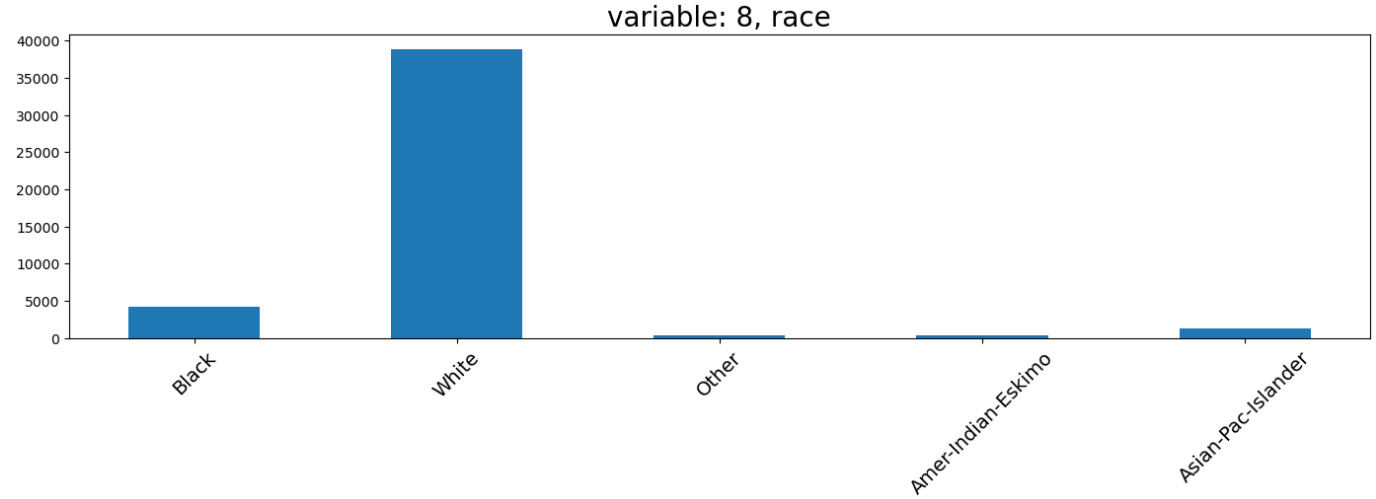

variable: 1, workclass
30000
25000
20000
15000
10000
5000
0
Private
Local-gov
Self-emp-not-inc
Federal-gov
State-gov
Self-emp-inc
Without-pay

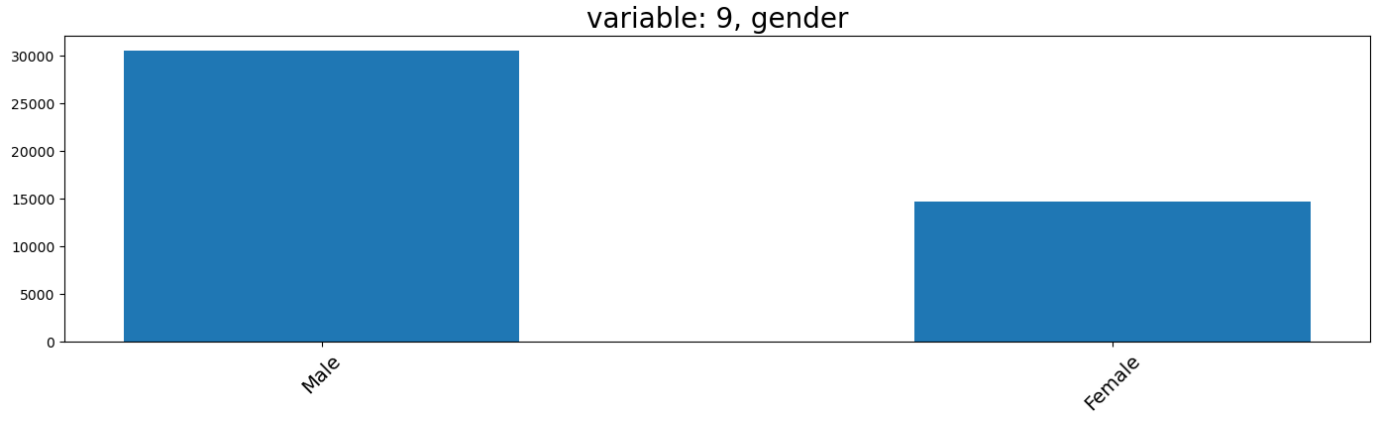

variable: 3, education
14000
12000
10000
8000
6000
4000
2000
0
11th
HS-grad
Assoc-acdm
Some-college
10th
Prof-school
7th-8th
Bachelors
Masters
5th-6th
Assoc-voc
9th
Doctorate
12th
1st-4th
Preschool

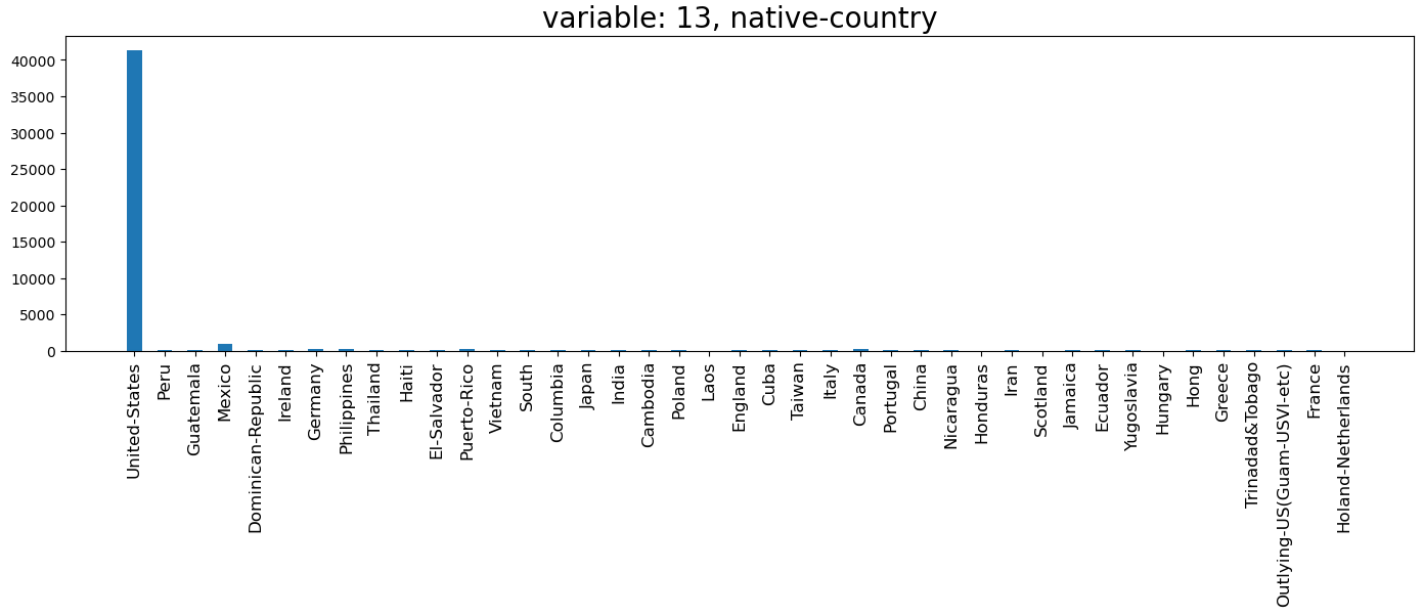

variable: 5, marital-status
20000
17500
15000
12500
10000
7500
5000
2500
0
Never-married
Married-civ-spouse
Widowed
Separated
Divorced
Married-spouse-absent
Married-AF-spouse

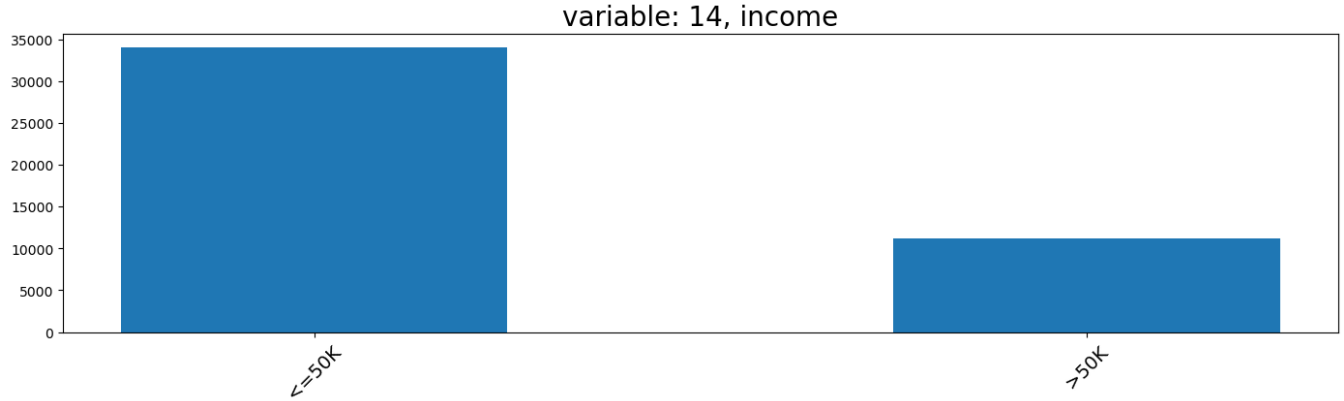

variable: 6, occupation
6000
5000
4000
3000
2000
1000
0
Machine-op-inspct
Farming-fishing
Protective-serv
Other-service
Prof-specialty
Craft-repair
Adm-clerical
Exec-managerial
Tech-support
Sales
Priv-house-serv
Transport-moving
Handlers-cleaners
Armed-Forces

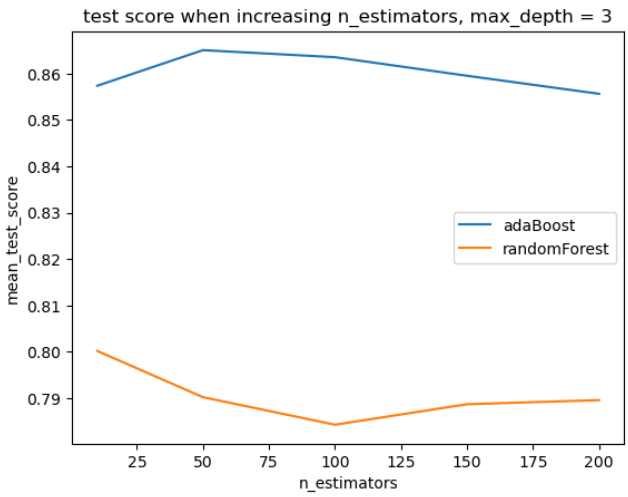

variable: 7, relationship
17500
15000
12500
10000
7500
5000
2500
0
Own-child
Husband
Not-in-family
Unmarried
Wife
Other-relative

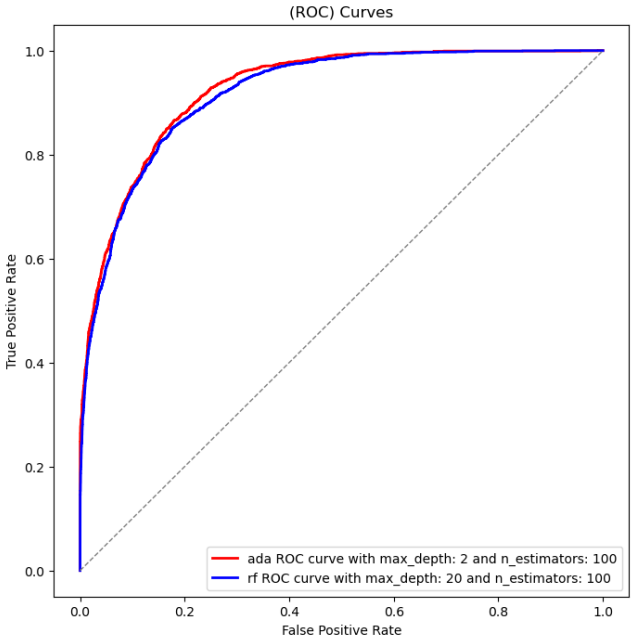

variable: 8, race
40000
35000
30000
25000
20000
15000
10000
5000
0
Black
White
Other
Amer-Indian-Eskimo
Asian-Pac-Islander

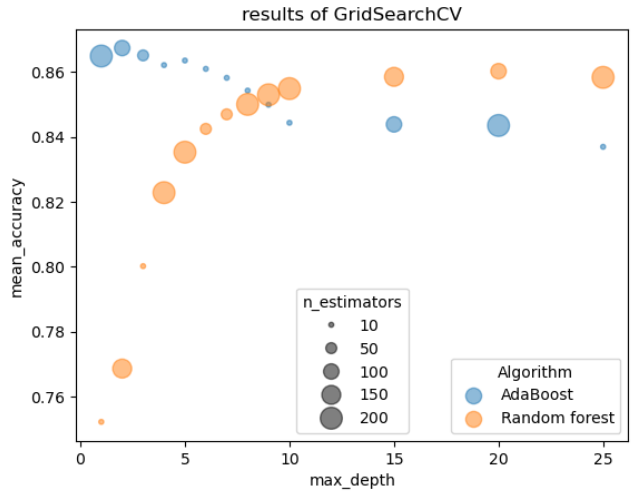

variable: 9, gender
30000
25000
20000
15000
10000
5000
0
Male
Female

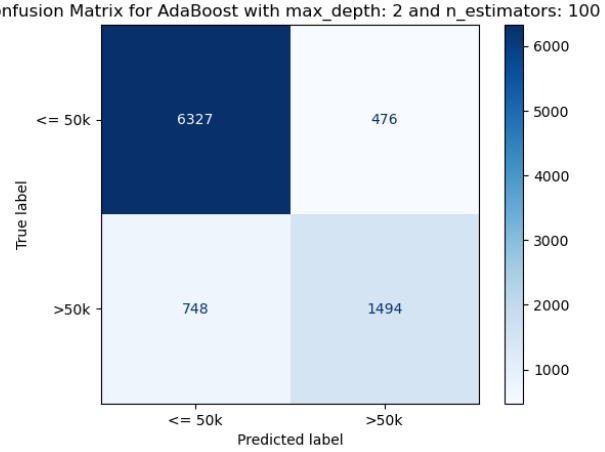

variable: 13, native-country
40000
35000
30000
25000
20000
15000
10000
5000
0
United-States
Peru
Guatemala
Mexico
Dominican-Republic
Ireland
Germany
Philippines
Thailand
Haiti
El-Salvador
Puerto-Rico
Vietnam
South
Columbia
Japan
India
Cambodia
Poland
Laos
England
Cuba
Taiwan
Italy
Canada
Portugal
China
Nicaragua
Honduras
Iran
Scotland
Jamaica
Ecuador
Yugoslavia
Hungary
Hong
Greece
Trinadad&Tobago
Outlying-US(Guam-USVI-etc)
France
Holand-Netherlands

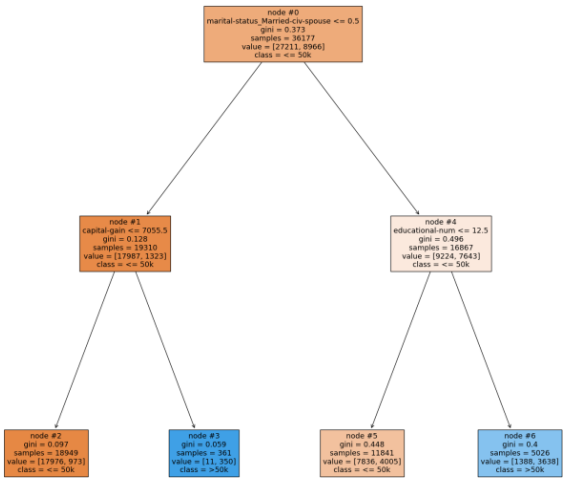

variable: 14, income
35000
30000
25000
20000
15000
10000
5000
0
<=50K
>50K



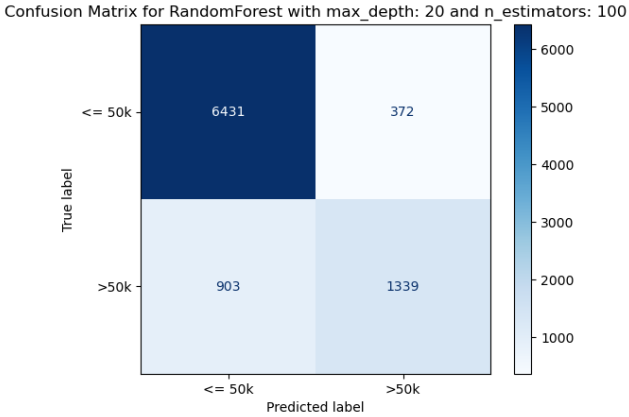


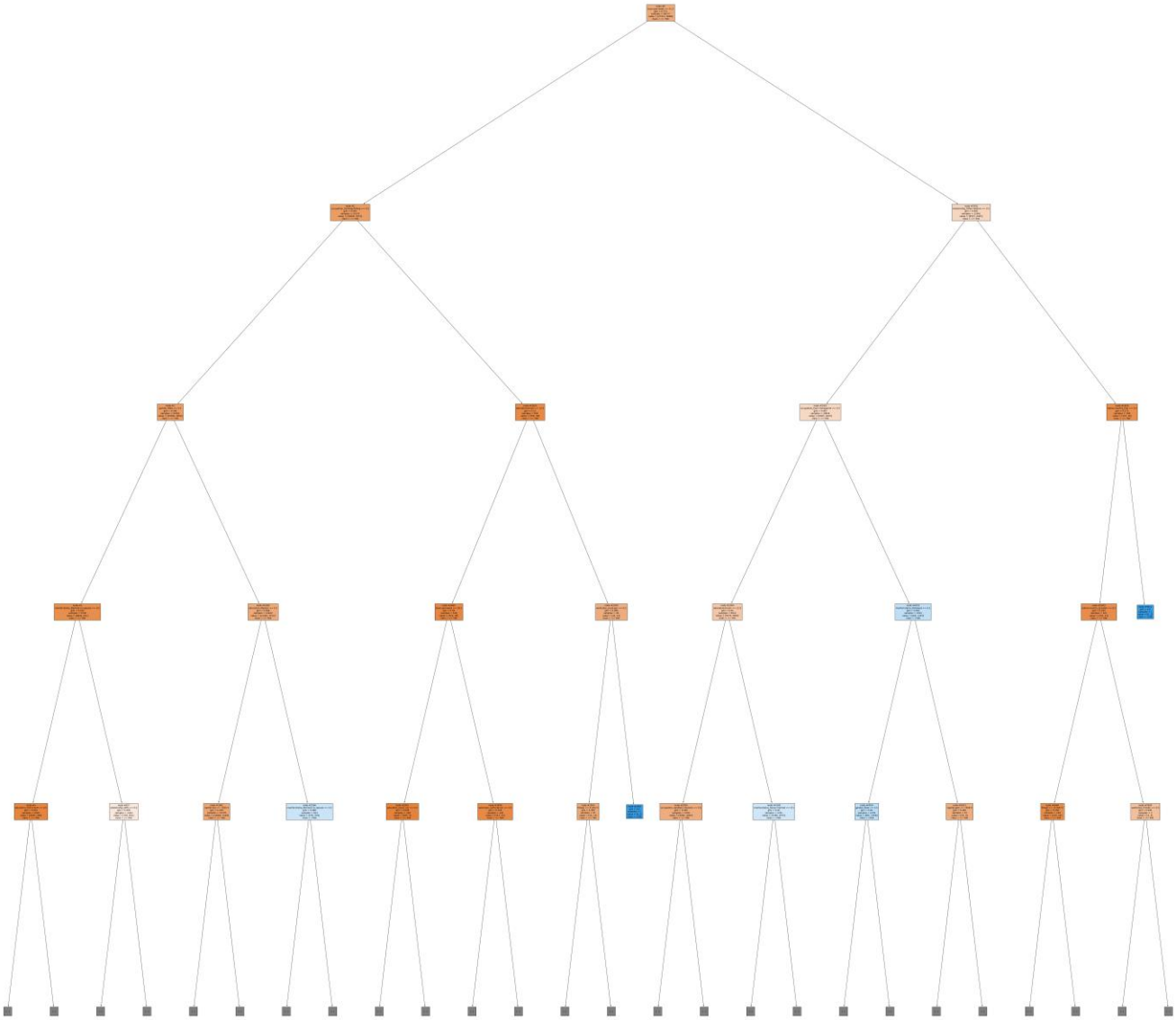






Best Decision Tree Visualization - AdaBoost with max_depth: 2 and n_estimators: 100





		randomForest	
		correct	wrong
adaBoost	correct	7520	301
	wrong	250	974