CS235 Fall 23 Project Mid Term Report: Income Classification using Census Data

AJIT SINGH, #1, NetID: asingh349
ADITYA GAMBHIR, #2, NetID: agamb031
ACHALA SHASHIDHARA PANDIT, #3, NetID: apand048
ANVITH REDDY NEMALI, #4, NetID: anema003

Additional Key Words and Phrases: Data Mining, Data Pre-processing, Random Forest Classification, Decision Tree Classification, AdaBoost, Artificial Neural Network

ACM Reference Format:
Ajit Singh, Aditya Gambhir, Achala Shashidhara Pandit, and Anvith Reddy Nemali. 2023. CS235 Fall 23 Project Mid Term Report: Income Classification using Census Data. In . ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 BRIEF SUMMARY OF THE STATE OF THE PROJECT

1.1 Current state

The dataset is unbalanced with class 0 (income <= 50K) being greater. The dataset however has some interesting correlations and the most interesting column of the dataset is the "fnlwgt" which after days of exploration and reading the dataset in and out, we figured out that it represented a multiplying factor that was to show how many people are likely to fall under a similar type of classification with similar features as the data set was not collected from all Americans, it helped give a overall general view of how to categorize all Americans. The data set gave out interesting correlations between income and education, income and capital gain/loss. While implementing off-the-shelf methods we also encountered a problem that will be discussed in the "Updates to the proposal" section 1.2 of the report. The current state of the project with the off-the-shelf implementation is available in the Google Colab link for implementation

1.2 Updates to the proposal

As per the proposal, Support Vector Machine (SVM) was one of the data mining techniques to be used. We implemented SVM after initial preprocessing, however, it was observed that during cross-validation and hyperparameter tuning, training the model took extremely long. On further exploring the reasoning behind this behavior we learned that the fit time scales at least quadratically with the number of samples and may be impractical beyond tens of thousands of samples. [1] Going forward implementing SVC from scratch and training with a large dataset such as ours does not seem feasible. As a result, we chose a different data mining algorithm, namely AdaBoost Classifier. Team members will be working with the following data mining techniques respectively.

- Ajit Singh: Random Forest Implementation

[1] https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

Manuscript submitted to ACM

- Aditya Gambhir: Artificial Neural Network approach
- Achala Shashidhara Pandit: AdaBoost Classifier method
- Anvith Reddy Nemali: Decision Trees Technique

2 PRELIMINARY RESULTS

Data preprocessing: Various preprocessing methods are carried out on the Adult Census Income Dataset, such as handling duplicate values, dealing with missing values, encoding the target variable ('Income'), Feature engineering, and selection.

As the total number of duplicates in the whole dataset was fewer than 25, it was dropped as it was inconsequential. Later, we scanned the dataset for missing values (NaN) and special characters like ['?']. Following this identification, we substituted the mode for the missing/special values in the 'native-country' and 'workclass' categories, as well as a new value 'Other' in the 'occupation' category. The target variable 'Income' was encoded where (<=50k) is 0 and (>50k) is 1.

Feature Engineering entailed encoding categorical features utilizing approaches such as One-Hot-Encoding, Label Encoding, and Frequency Encoding. Considering we're working with ANN and SVC, the numerical features have been scaled using standard and min-max scaling. Additionally, we've dropped irrelevant features, in our case education was dropped as it was redundant, and a numerical representation of the same, called education-num was already present,

The dataset is then split into two parts, namely training and testing, on which we'll run each of our models. The model's performance is compared using a variety of performance metrics such as accuracy, precision, f1-score, recall, and a confusion matrix. These indicators will be used to make the Selection .

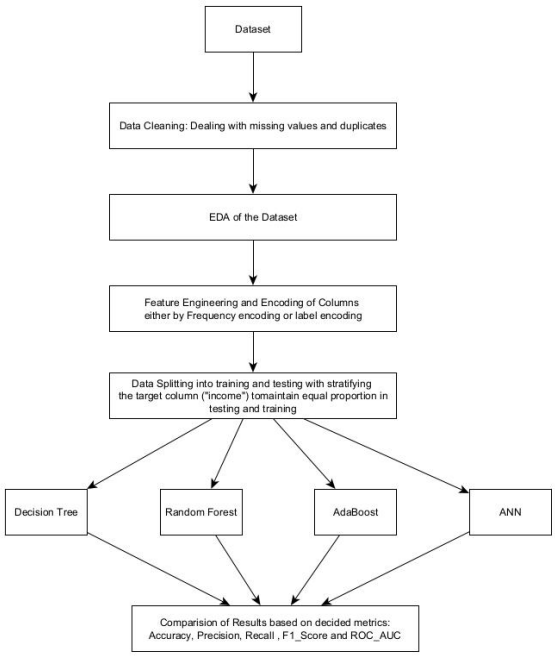


Fig. 1. Caption

2.1 Random Forest Classification

2.1.1 ***Introduction:*** A random forest (RF) is a meta-estimator that fits several decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting. The sub-sample size is controlled with the max samples parameter if bootstrap = True (default), otherwise, the whole dataset is used to build each tree.

2.1.2 ***Baseline Model:*** Off-the-shelf baseline model: The library used to import the baseline model is 'sklearn.ensemble'. RF is a bagging ensemble method of decision trees. The model is fitted with the training and testing data and the following were the results on different predefined metrics:

- Accuracy: 84.95 %
- Precision: 71.65 %
- Recall: 61.43 %
- F1 Score: 66.15 %
- ROC AUC: 89.66 %

The feature importance was also derived for the base model. Feature importance shows which feature has the most influence in deriving the result/ predicted classifier result. The following are the top 5 feature classifications:

Table 1. Feature Importance

Feature	Importance
fnlwgt	0.23530280519077287
education-num	0.1247166693670773
capital-gain	0.11665151176241312
relationship	0.1026755730504701
hours-per-week	0.09222174834296967

Table 2. Class-Based Accuracy

Class	Accuracy
Class 0	92.32 %
Class 1	61.87 %

The cross-validation accuracy for the model is 85%

Following the base model it was determined that hyper tuning of the parameters of RF should be done to determine if it would help increase the performance of the model.

2.1.3 ***Hyperparameter Tuning.*** The RF model has a lot of hyperparameters to fine-tune such as n_estimators, max_depth, min_samples_split, max_features, bootstrap, max_samples, etc.

Choosing the parameters to fine-tune was a difficult task. For this understanding how the chosen parameters affect the performance of the RF is important. The following parameters were chosen: Max_depth: max depth of the trees

Max_features: no of features to consider for the best split Min_samples_split: min sample required to split an internal node Bootstrap: boolean value that dictates whether to bootstrap samples or not criterion: The function used to measure the quality of a split. It can be "gini" for Gini impurity or "entropy" for information gain.

To tune the parameter and to select the best tuning, the Random Search tuning method was used from the sklearn library. In this, the parameter is defined and a range is given the random search algorithm tries to randomize the selection process of the tuning and chooses the best possible combination of tuning for each parameter.

The above parameters were tuned and the best possible combination was: 'bootstrap': True, 'criterion': 'gini', 'max_depth': 5, 'max_features': 9, 'min_samples_split': 28 The decision to use random search over grid search was based on computational limitations as well as time to execute the algorithm. Moreover, random search is more efficient and flexible. The reported results were as follows:

- Accuracy: 85.12 %
- Precision: 78.62 %
- Recall: 52.04 %
- F1 Score: 62.41 %
- ROC AUC: 89.95 %

The choice of the hyperparameters in this case was determined by factors like computational resources, and influence on the tree moreover these are some of the most commonly tuned hyperparameters, and tuning these for the given data set has shown a small increase in accuracy and a significant increase in precision however recall power of the model reduces.

2.2 Artificial Neural Network

2.2.1 ***Introduction:*** Artificial Neural Networks (ANNs) draw inspiration from the structure and functionality of the human brain. They are comprised of interconnected nodes, often referred to as artificial neurons or perceptrons, organized into layers. ANNs serve multiple purposes, including tasks such as pattern recognition, classification, and regression. In the context of this analysis, a specific dataset designed for binary classification was used, and an off-the-shelf model was imported from the TensorFlow[2] library.

2.2.2 ***Baseline Model:*** Initially, a baseline neural network model was trained without hyperparameter tuning. The model consisted of one hidden layer with 64 units and a ReLU activation function in the input layer, 32 units with a ReLU activation function in the hidden layer, and one unit with a sigmoid activation function in the output layer. The training process employed the Adam optimizer and binary cross-entropy loss function, resulting in the following metrics:

- Precision: 0.7459
- Recall: 0.5619
- Accuracy: 0.8493
- F1-Score: 0.6409
- ROC-AUC: 0.9045

2.2.3 ***Hyperparameter Tuning:*** Customizing an ANN involves fine-tuning hyperparameters as the model's performance significantly relies on finding the right combination of these parameters. To pinpoint the optimal hyperparameters,

[2] https://www.tensorflow.org/api_docs/python/tf/keras/

a 'RandomSearch' tuner from the 'keras_tuner'[3] library was employed with the primary goal of maximizing validation accuracy. This process led to the discovery of the most suitable hyperparameters:

- Number of Hidden Layers: 1
- Units in Hidden Layer 1: 192
- Learning Rate: 0.01

Alongside these hyperparameters, the hidden layer incorporated a ReLU activation function, and the output layer featured a sigmoid activation function. The optimization process utilized the Adam optimizer and binary cross-entropy loss function.

2.2.4 ***Cross-Validation:*** To ensure the model's robustness and mitigate overfitting, a k-fold cross-validation approach was implemented. 'StratifiedKFold' was used to split the data into 10 equally sized folds. Training took place on K-1 folds, while validation occurred on the remaining fold for 20 epochs. Model performance was assessed using various metrics, including precision, recall, accuracy, F1-score, ROC-AUC, and class-specific accuracies. These metrics were aggregated to gauge the model's overall performance.

2.2.5 ***Evaluation Metrics:*** Upon evaluating the model's performance on the testing set, the following results were obtained:

- Precision: 0.7955
- Recall: 0.4944
- Accuracy: 0.8485
- F1-Score: 0.6098
- ROC-AUC: 0.9025

Apart from these metrics class-based accuracy was also calculated:

Table 3. Class-Based Accuracy

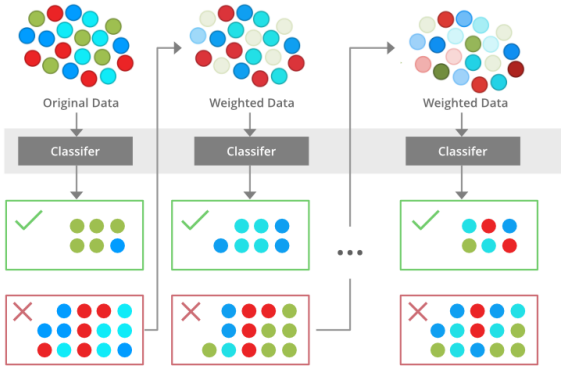
Class	Accuracy
Class 0	0.9440
Class 1	0.5518

2.2.6 ***Conclusion:*** These metrics shed light on the model's effectiveness in accurately classifying individuals based on their income levels. Even though fine-tuning the hyperparameters didn't have a significant effect on the model performance, the best possible permutation of hyperparameters was found. Precision reflects the proportion of true positive predictions, recall measures the model's ability to identify actual positive cases, accuracy gauges the overall correctness of predictions, the F1-score balances precision and recall, and ROC-AUC assesses the model's capacity to distinguish between classes. In the context of this analysis, a high ROC-AUC score signifies the model's proficiency in classifying individuals by their income levels. However, the moderately lower recall suggests potential room for improvement in identifying high-income individuals.

[3] https://keras.io/keras_tuner/

2.3 Adaptive Boosting Classification

2.3.1 ***Introduction:*** Adaptive Boosting, also called AdaBoost, is a supervised ensemble learning technique. The philosophy behind ensemble learning techniques is to use the advantages of many models, i.e. a single model may not perform well, however many models together may be able to predict better. In the case of boosting algorithms, an initial classifier is trained and tested. The next classifier takes up all the samples that were mispredicted by the first classifier along with newer samples and trained. 'N' number of such classifiers are trained as shown in Fig that are used together while predicting the class of a new sample. In the case of AdaBoost, each of these classifiers is a decision tree of height one and is called a stump. Every time samples are misclassified and passed on to the next classifier a higher weight is given to these samples to provide greater emphasis to them. Some stumps can also be better than others, hence a significance value is calculated for each stump. When a new sample has to be classified, every stump predicts a certain class and a weighted average based on significance is taken to make a decision.



[4]

Fig. 2. Boosting Technique

2.3.2 ***Baseline Model:*** The initial model was trained using off-the-shelf sklearn implementation. sklearn.ensemble.AdaBoostClassifier [5] fits additional copies of the classifier on the same dataset but alters weights of incorrectly classified instances in subsequent classifiers as discussed above. The baseline model was trained with default parameters 50 n_estimators, 1.0 learning rate, and SAMME.R algorithm.

- n_estimators: indicates the number of weak learners used in the ensemble technique
- learning rate: weight applied to each classifier at every boosting iteration. A higher learning rate increases the contribution of each classifier.
- SAMME.R/SAMME: With SAMME.R algorithm, the final weight is indicated by the probability of a sample belonging to a class. On the other hand, SAMME gives a discrete value like 0 or 1 classification.

With the default parameters model, the performance observed is as follows.

- Precision: 0.7633
- Recall: 0.5824
- Accuracy: 0.8568
- F1-Score: 0.6607

[5] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.htmlsklearn.ensemble.AdaBoostClassifier

2.3.3 ***Hyperparameter Tuning:*** Hyperparameters for AdaBoost include several weak learners, the learning rate, and the algorithm used to decide the final classification. RandomizedSearchCV [6] is another tool provided by sklearn that helps to identify good values for hyperparameters. Given a set or range of hyperparameters, it randomly selects a subset of these values and evaluates the model based on them to identify a good set of hyperparameters. In this case, 1000 estimators with a learning rate of 1.0 on the SAMME algorithm were chosen. With these parameters, another model was trained and the below performance was observed.

- Precision: 0.7681
- Recall: 0.5834
- Accuracy: 0.8581
- F1-Score: 0.6631

2.3.4 ***Cross-Validation:*** To ensure the model performs well under most circumstances and does not over-fit on data k-fold cross-validation was implemented. 10-fold cross-validation was implemented on the model resulting in an accuracy of 86.02%.

2.3.5 ***Evauation Metrics:*** After training models with and without hyperparameter tuning various performance metrics such as accuracy, recall, precision, and F1 score were noted.

The class-based accuracy of the model is 94.32% for income below $50K and 58.24% for income above $50K. Further, since each of the weak learners are decision tree, we can extract the feature importance of the model. The top 5 features are listed in Table 4.

Table 4. AdaBoost Classification Feature Importance

Feature	Importance
capital-gain	0.330475922621524
capital-loss	0.19269425241917895
relationship	0.17923694223045872
education-num	0.08116574772140728
occupation	0.07893632226322883

2.3.6 ***Conclusion:*** Firstly, the performance metrics of the models with and without hyperparameter tuning are comparable. In both scenarios, an accuracy of about 85% is observed. However, there exists low recall and precision scores. It is inferred that around 76% of the samples that are predicted as income above $50K are correctly identified but fail to identify all data points whose income is above $50K resulting in a very low recall score of 58%. Additionally, high accuracy can be seen in identifying samples of income below $50K in contrast to low accuracy in identifying samples of income above $50K. This can be due to the unequal distribution of training samples. Further, extracting feature importance helps understand which features played a crucial role. It is observed that capital gain and loss, relationship of dependents, educational qualification, and occupation have higher priority which is also intuitive from the initial data exploration. Going forward since there is not a large increase in performance from altering hyperparameters while the number of estimators changes from 50 to 1000 resulting in a huge increase in computation, it would be more feasible to implement the algorithm by scratch based on default sklearn parameters.

[6] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.htmlsklearn.ensemble.AdaBoostClassifier

2.4 Decision Trees Classification

2.4.1 ***Introduction:*** A decision tree classifier is a tree-structured machine learning model that can be used for classification as well as regression. It functions by recursively splitting data into subsets based on the most relevant features and then classifying the data utilizing a tree structure. Each internal node of the decision tree represents an attribute test, each branch represents a test result, and each leaf node stores a class label. Although the decision tree is very capable it can be prone to overfitting and may lead to the creation of biased trees, especially if the data is noisy or imbalanced.

2.4.2 ***Baseline Model:*** The library used to implement the model was from the `sklearn` library, specifically the `DecisionTreeClassifier` class. This class has several hyperparameters, such as `max_depth`, `criterion`, `max_features`, `splitter`, `min_samples_leaf`, etc., that can be tuned to improve the performance of the Decision Tree. The `DecisionTreeClassifier` algorithm was the base classifier that was used with the training and testing data, and initially, all the parameters belonging to the base classifier were set to their default values. These default values resulted in the following performance results

- Accuracy: 0.8134
- Precision: 0.6082
- Recall: 0.6197
- F1-Score: 0.6139
- ROC AUC: 0.8963

The accuracy measures the number of samples that were correctly identified from the total number of samples, the decision tree has an accuracy of 81.34 percent. The precision of 0.6082 essentially means that it correctly predicted 60.82 percent of the samples that represent income levels above 50k. Recall of 0.6197 indicates that 61.97 percent of the correctly identified samples have incomes more than 50k. An F1 score of 0.6139 indicates that the classifier has a balanced performance between precision and recall. ROC-AUC measures a model's ability to differentiate between both positive and negative classes, with greater values suggesting better discrimination performance.

The technique of feature importance is used to discover which features in a dataset are most crucial in predicting the target variable. Here are the top 5 feature importances determined for the base model:

Table 5. Decision Tree Classification Feature Importance

Feature	Importance
fnlwgt	0.200955
relationship	0.200154
education-num	0.125126
age	0.119426
capital-gain	0.112687

2.4.3 ***Hyperparameter Tuning:*** Subsequently, hyperparameter tuning was performed to understand how each chosen parameter impacts the model performance. 'criterion,' 'splitter,' 'max_depth,' 'min_samples_split,' and 'min_samples_leaf. were the parameters that were used. These parameters have been selected as they are essential for the decision tree's behavior, particularly when it pertains to over-fitting, impurity assessment, and granularity of splits.

'criterion': The metric, usually 'entropy' for information gain or 'gini' for Gini impurity, that is employed to evaluate the quality of a split in a decision tree.

'splitter': The method for choosing the most advantageous characteristic to divide the data into two categories: 'random' (chosen at random) or 'best' (chosen based on a quality metric).

'max_depth': The decision tree's maximum growth depth, which restricts its complexity and overfitting potential.

'min_samples_split': The bare minimum of samples necessary to divide a node in a tree-building operation.

'min_samples_leaf': This parameter controls the granularity of splits by indicating the minimum amount of samples needed to form a leaf node in the decision tree.

For hyperparameter tuning, we used Random Search because it enables us to explore a wide range of hyperparameter combinations within a shorter timeframe, which is particularly useful when computational resources are limited. In Random Search, the technique randomly samples various combinations of hyperparameter values to discover the optimal set for a machine learning model.

At the time of writing/running, the best hyperparameters obtained were:

Best Hyperparameters:

{ 'splitter': 'best', 'min_samples_split': 3, 'min_samples_leaf': 5, 'max_depth': 7, 'criterion': 'gini' }

The evaluation metrics after using these hyperparameters are as follows:

- Accuracy: 85.26 %
- Precision: 78.99 %
- Recall: 52.37 %
- F1-Score: 62.99 %
- ROC AUC: 89.63 %

2.4.4 ***Conclusion:*** In this study, a Decision Tree Classifier was used to classify income levels, with an initial base model performing fairly. Pre-processing methods were used, such as handling duplicates and encoding category characteristics. However, following thorough hyperparameter adjustment, the model's performance improved, optimizing parameters such as 'criterion,' 'splitter,' 'max_depth,' 'min_samples_split,' and 'min_samples_leaf.' The utilization of these hyperparameters resulted in an accuracy of 85.26 percent with improved precision and a good F1 score. Feature importance analysis highlighted the important categories such as 'fnlwgt,' 'relationship,' 'education-num,' 'age,' and 'capital-gain.' Decision Trees have potential as good classifiers, particularly when fine-tuned to handle overfitting and impurity issues.