Report as part of Big Data Final Project Submission

Donor Classification Using AWS Sagemaker

Submitted By Team 3-

Alexander Ilyin
Ashish Gupta
Bhawna Gupta

Table of Contents

Problem Description

NGOs are now more than ever playing an extremely crucial role in overseas funding and development ventures in both developing and developed economies. Statiscally, it has been seen that high-income individuals have been proactively donating to charitable organizations, which accounts for 70% of all the donations that are made to these NGOs. The underlying issue with this regard is that NGOs are not politically independent and receive paybacks from various governmental institutions, thus leading to a skewed system. We intend to use the capabilities offered by data science to improve this situation.

Our objective in this project is to identify if an individual earns more than $50k(high income bracket) and request optimum donation amounts based on their individual incomes. Our analysis can provide an efficient way for NGOs to have a substantial amount of donation, by just targeting a few but ‘right’ individuals.

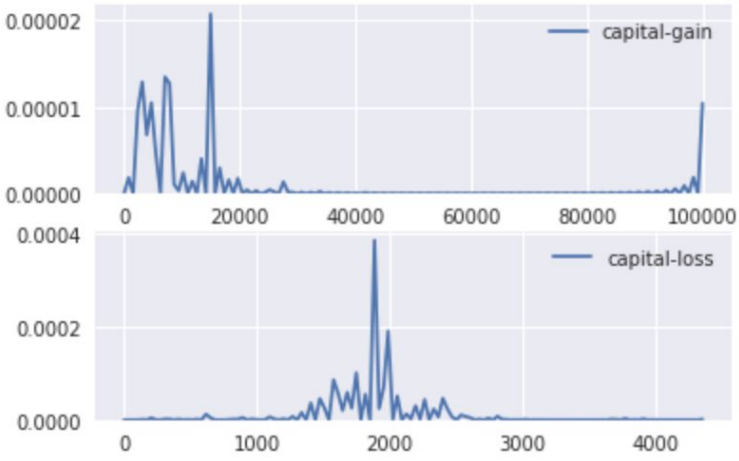
Data Preparation

In order to achieve the objective, we performed a classic scenario of a classification task. We obtained a large dataset of individual incomes with explanatory variables listed in the below table. This dataset is fetched from the UCI machine learning repository https://archive.ics.uci.edu/ml/datasets/Adult.

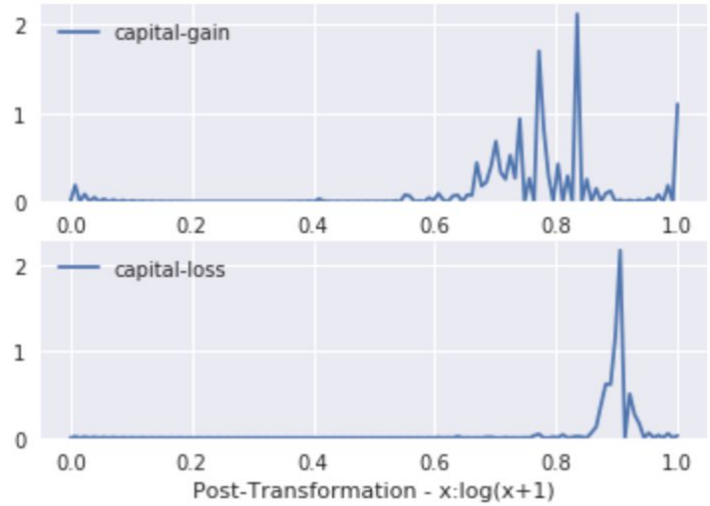
Feature	Description	Dtype	Missing Values
age	Age	int64	NA
workclass	Working Class	str	1836
education_level	Level of Education	str	NA
education-num	Number of educational years	int64	NA
marital-status	Marital status	str	NA
occupation	Work Occupation	str	1843
relationship	Relationship Status	str	NA
race	Race	str	NA
sex	sex	str	NA
capital-gain	Monetary Capital Gains	int64	NA
capital-loss	Monetary Capital Losses	int64	NA
hours-per-week	Average Hours Per Week Worked	int64	NA
native-country	Native Country	str	583
income	Income Class	str	NA

A. Exploratory analysis

For the features capital gain and capital loss, we notice a fluctuating curve with random spikes at certain positions. Some capital gain data values are way beyond the mean value forming extreme outliers.



To prevent this from impeding the classification modeling process, we take a log transformation on these features. The issue here is that multiple data points have zero values in these features, and logarithm of 0 is undefined. Hence, we increment the value slightly and then obtain the log transformation as shown below to restrict the values between (0,1).



B. Dealing with missing values

Upon viewing our data, we found that we have multiple missing values in a few features represented by the string '?'. The count of these missing values is shown in the table

above. We notice that most of the features are multi-valued categorical features and around 5 features are numeric. Overall, these variables can be intuitively interpreted to cause an effect in an individual's income level. First glance at the dataset helps us conclude that certain features require transformations and cleaning.

Initially, we hoped to find a statistical or machine learning based approach to impute these missing values. For example, using the mean of the column or using a classifier to impute values will allow us to keep these entries instead of deleting them.

However, using the mean of a column with categorical values is not possible. While using a classifier was still possible for these variables, the three columns that include missing values are workclass, occupation and native country. These variables are related to a person's demographics, and would not make much sense to predict. The fact that we only have around 40,000 rows of data makes it unlikely that a classification would produce accurate results for these columns. For these reasons, we decided to cut our losses in terms of losing data points and removed all missing values from our dataset.

C. Scaling numeric features

Algorithms like random forest and gradient tree boosting work best if the numeric values are standardised to prevent scaling issues when training the model. This is done by subtracting the mean and dividing by the standard deviation.

	age	education-num	capital-gain	capital-loss	hours-per-week
count	24129.000000	24129.000000	24129.000000	24129.000000	24129.000000
mean	0.293319	0.607313	0.064519	0.043073	0.407243
std	0.180134	0.170278	0.214638	0.191790	0.122587
min	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.150685	0.533333	0.000000	0.000000	0.397959
50%	0.273973	0.600000	0.000000	0.000000	0.397959
75%	0.410959	0.733333	0.000000	0.000000	0.448980
max	1.000000	1.000000	1.000000	1.000000	1.000000

D. One - Hot Encoding categorical features

This follows the same principle as for scaling of numeric features to improve model performance where interactions and non-linearities are accounted for. The multi-valued categorical features are therefore one-hot encoded.

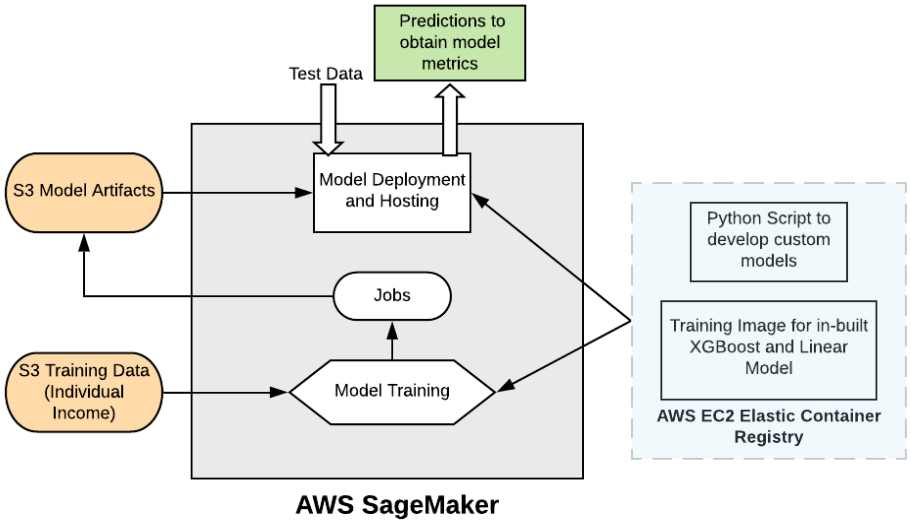
E. Train Test Split

To follow the guidelines for machine learning model building, we split out data in a training and test set. We used a split of 80/20, with 80% of the data going to training and 20% going to the test set. In order for consistency among our models, it was important to use the same data splits as well as using standardized variables for all three models.

Approaches

A. Architecture

AWS Sagemaker turned out to be our ideal choice since its offerings provided for a collectively exhaustive range for all our implementations. We used AWS S3 storage solution to load our dataset and created a notebook instance using the conda environment to run python and fetch the data from S3. We used the Linear model and XGBoost in-built training images offered by SageMaker to train the classification model and store the results in artifacts. We also developed a random forest implementation using the sklearn functionality and a separate script to feed it as an entry point into SageMaker. The models were deployed and the test data was used to make predictions and compare different models.

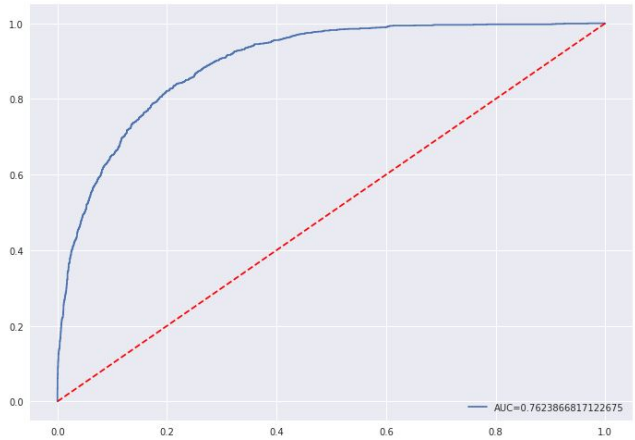


B. Linear Model Implementation

While AWS/Sagemaker does not have a dedicated logistic regression model, it is possible to create a logit model using the linear-learner function. What differentiates the logistic regression in the function call is the "predictor_type" argument. Setting this to "binary_classifier" will cause the linear learner to predict for labels. The advantage of the linear learner model is that its prediction output can be set as both the probability and predicted label. This option was not available for both Random Forest and XGBoost, and our workaround will be discussed further in the report.

The steps to create a linear learner are identical to other AWS Sagemaker, however, the amount of possible hyperparameters is quite limited for this model. One important hyperparameter that separates linear learners from other models is the "L1" regularization parameter. Using this hyperparameter, we could penalize our model for excessive complexity and deal with overfitting. This was especially helpful since we were using a large amount of features after one-hot encoding. Another hyperparameter we used was "learning rate", which controls how aggressively our function attempts to minimize our loss function. Since the optimization process of a linear model involves the use of gradient descent, the learning rate controls the size of the "steps" that the linear model takes when computing the iterative gradient of the loss function. A higher learning rate can cause the gradient descent to possibly miss the global minimum of the loss function, or converge to a local minimum before possibly finding the global minimum (which is the goal). Finally, we adjusted positive sample weights (increasing weight of samples with income >50k) and whether or not our model used bias.

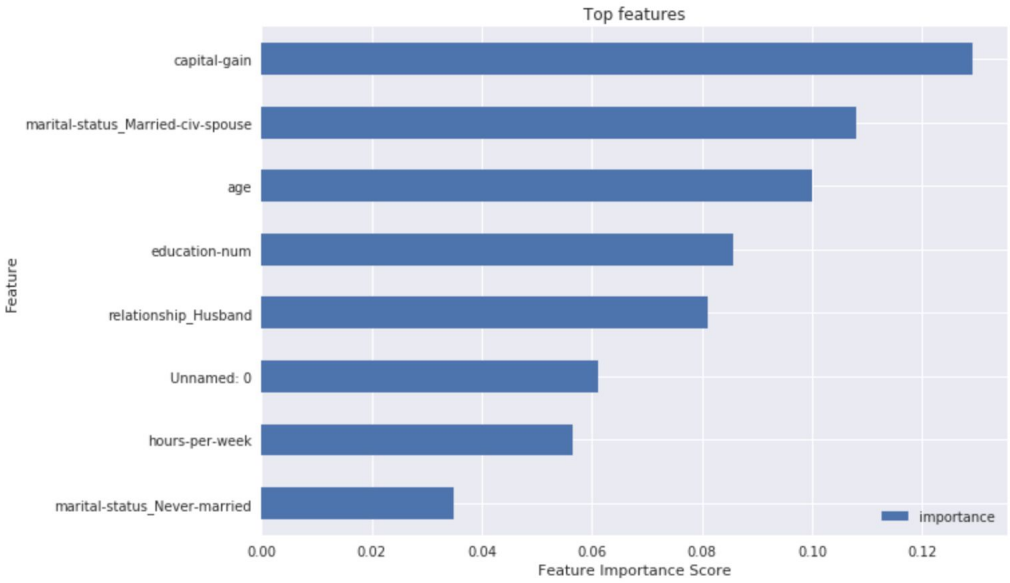
Even with adjusting these hyperparameters, this model was the worst performing out of the three we chose, based on all three metrics. This could be due to a multitude of factors, the most likely of these being that the other models were able to pick up non-linear relationships among some of the predictor variables which vastly improved their performance. Shown below is the Receiver Operator Curve(ROC):



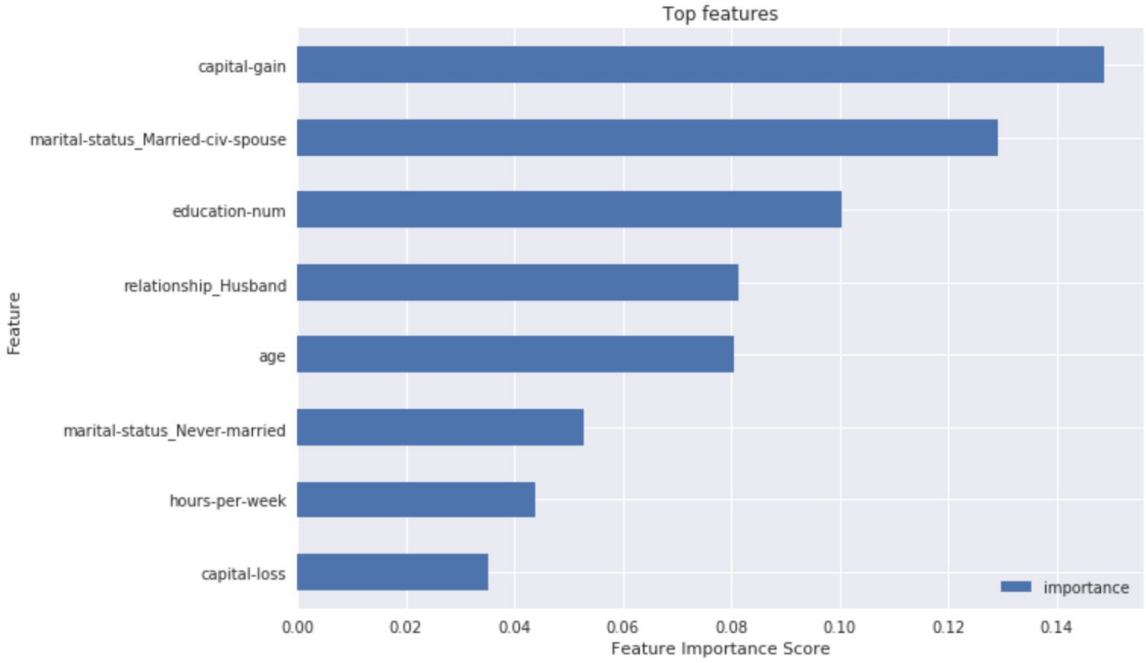
C. Random Forest Implementation

Since random forest is not offered as a training image directly by AWS SageMaker, we create a custom script using the Sklearn package and setting static hyperparameters to run the model. This is fed as an entry point to the sklearn model function in the SageMaker training instance to create a training job. This is then saved as an artifact. We extract the saved models to obtain the probability predictions on the test data to obtain various metrics like auc, precision and recall. The AUC shows a high bend in the curve exposing daylight between the blue and the trivial red curve, thus suggesting that this is a good model.

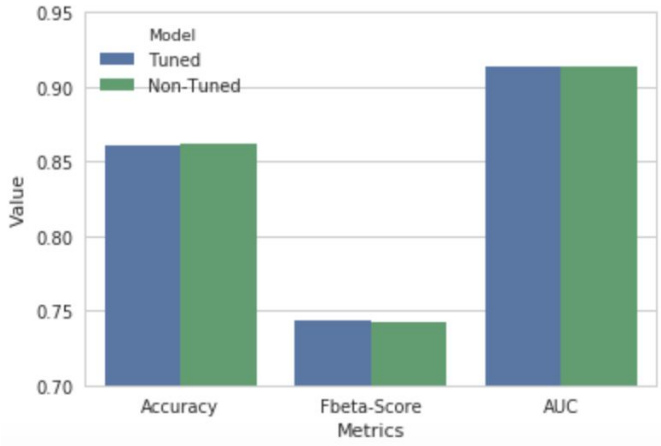
We also obtain the top feature importances to identify the top attributes in the model performance. This is shown below.



We notice that capital gain, marital status being a spouse and age are the top three features. The importance is distributed with a constant decrease in the top 10 features. We then tune the hyperparameters to improve the model performance. We use a larger set of trees to train the model and with higher minimum nodes in the leaf positions. We get a slightly different feature importance. Capital gains still is on the top followed by marital status, but age drops to a lower position and is overtaken by features like education years and if the individual is a husband.



We compare the metrics between the tuned and the hyperparameter tuned model and notice that there is a slight improvement in the auc and f1-score for the tuned model.



D. XGBoost Implementation

The XGBoost (eXtreme Gradient Boosting) is a popular and efficient open-source implementation of the gradient boosted trees algorithm. While implementing XGBoost, we took the advantage of the dedicated in-built feature provided by SageMaker. We chose to use SageMaker because of the advantages we get:

- Easy deployment and managed model hosting
- Out-of-the-box distributed training
- XGBoost Instance Weighted Training
- Spark integration with the Spark SDK

Continuing with our variables, we first train our model using the offerings provided by AWS, by creating a training job. The training job includes the following information - The URL of the Amazon Simple Storage Service (Amazon S3) bucket where we have stored the training data, The compute resources that we want Amazon SageMaker to use for model training which are basically ML compute instances that are managed by Amazon SageMaker, the URL of the S3 bucket where we want to store the output of the job, the Amazon Elastic Container Registry path where the training code is stored. When we create a training job with the API, Amazon SageMaker replicates the entire dataset on ML compute instances by default.

After we create the training job, Amazon SageMaker launches the ML EC2-compute instance and uses the training code and the training dataset to train the model. It saves the resulting model artifacts and other output in the S3 bucket we specified for that purpose. Once we get our trained model, we use the hyperparameter tuning that SageMaker provides that we can tune to improve model performance. Here, we set values for some of the most commonly tuned hyperparameters.

Our best model is obtained using the parameters: eta= 0.2, gamma = 3, max_depth=5 and min_child_weight=6.

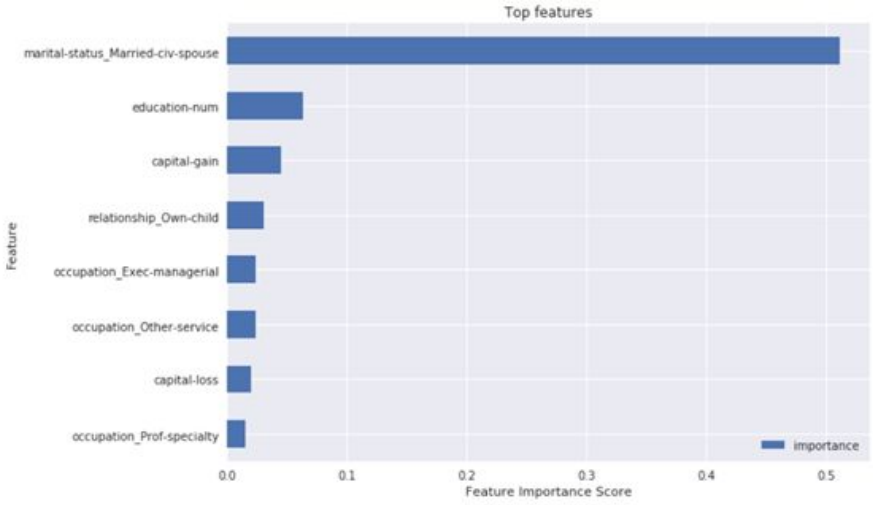
Step size shrinkage is used in updates to prevent overfitting. After each boosting step, we can directly get the weights of new features. The eta parameter actually shrinks the feature weights to make the boosting process more conservative.

Gamma is the minimum loss reduction required to make a further partition on a leaf node of the tree. The larger, the more conservative the algorithm is.

Min_child_weight is the minimum sum of instance weight (hessian) needed in a child. If the tree partition step results in a leaf node with the sum of instance weight less than min_child_weight, the building process gives up further partitioning.

Max_depth is the maximum depth of a tree. Increasing this value will make the model more complex and more likely to overfit. 0 is only accepted in lossguided growing policy when tree_method is set as ‘hist’ and it indicates no limit on depth.

On plotting the feature importance, we see that the marital-status has highest importance. Followed by education and capital gain. Although the top features from XGBoost are similar to that of Random Forest, we see a substantial difference in feature importance of top 2 features in XGB. This is because the XGB runs on Boosting where it identifies the most important variable and focuses its learning improvement on this feature. Whereas, in RF the distribution is equally distributed as it uses bagging/bootstrap sampling for its learning process and this causes more distributed feature importance.



Challenges and solutions

A. Deployed models do not allow for the predicted probabilities

Since we were using a Sklearn function as an entry point for Sagemaker, we could not use the prediction capabilities of Sagemaker, which allow for the output of both the probability and predicted label for each data point. In order to get around this problem, we used the Sklearn’s joblib functionality in order to load our tuned model as a Sklearn model. After loading the model as a Sklearn model we were able to use the predict_proba function in Sklearn to get the probabilities we desired.

B. Poor initial model performance

Upon our first training iteration our tree based models, we found that these models were performing far below expectations. It is generally accepted that tree based models will perform better when trained on standardized variables. For this reason, we introduced the Minmaxscaler, which standardized our variables to a range from 0 to 1.

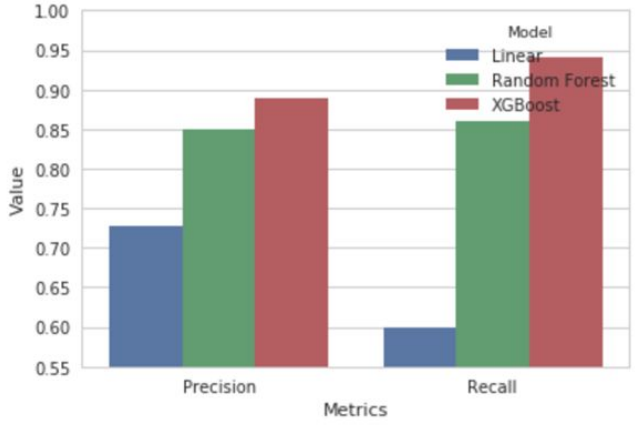
C. Poor logistic regression performance/complications in extracting feature importance

When comparing the three models we created, we found that the logistic regression was far behind in terms of the four metrics we selected. We hypothesize that this could be due to the fact that our tree based models were able to pick up non-linear relationships among our dependent variables. We also felt that the fact that we could not find a way to add interactions to the linear learner model hindered its performance. Also, since the goal of our project was to determine the importance of the different features in our dataset, we hoped to use feature importance in our final model. However, we could not find a feasible approach to extracting feature importance from our logistic regression. Since we were not able to find workarounds to these challenges, we simply excluded the logistic regression from our final model decision.

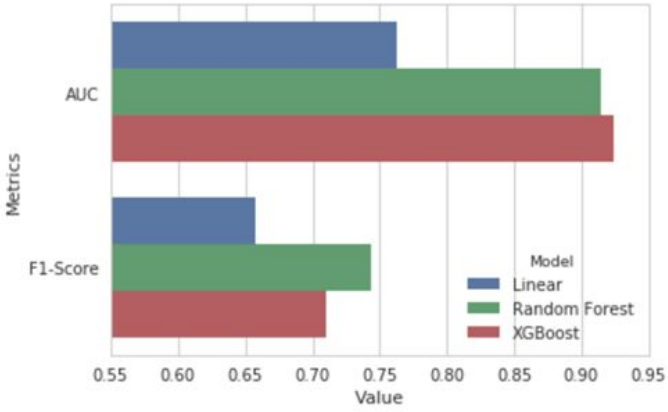
Analysis results

A. Model Comparison

After obtaining all the three model predictions, we now compare different models based on various performance metrics.

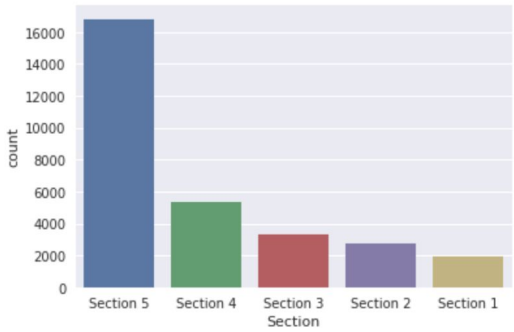


A side-by-side comparison of all the models based on precision and recall is shown alongside. Precision is basically defined as out of total positive predictions, how many are correct. Recall is the ratio of the correct predictions to the total actual positives. Our focus is having a high recall since we want to decrease the false negative predictions. Essentially, we do not want to lose out on an individual who has a high net worth and willing to donate more. Therefore it is fair to say that XGboost is the winner. We also compare our models on the basis of AUC on the test data. We see that it is pretty high compared to the trivial mean model. On the right side is side by side comparison of AUC and F1. Even though F1 is a little high for random forest, our main metric to decide the best model was AUC, and hence we chose XGBoost to make the final predictions of probabilities on the data.

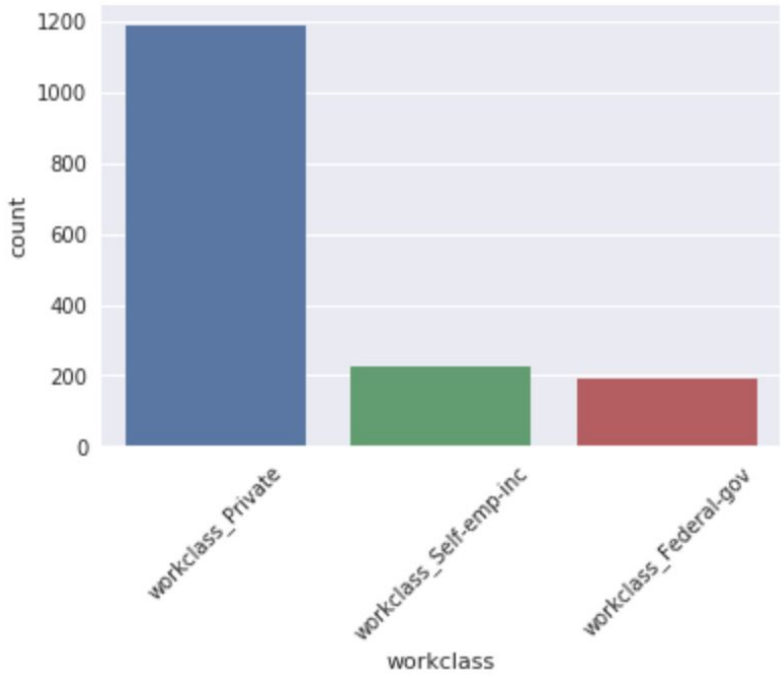


Insights gained

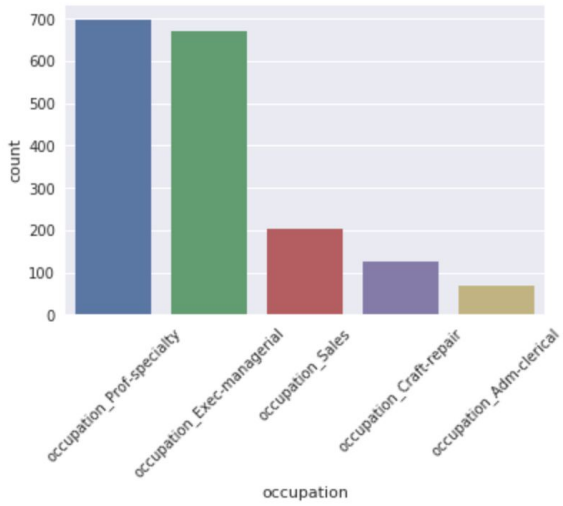
Once we obtain the best model predictions and the probabilities, We divide these probabilities into 5 bins named section 5-1. Section 1 having the highest probabilities and individuals with the highest income. The chart below is the distribution of individuals per section throughout the data.

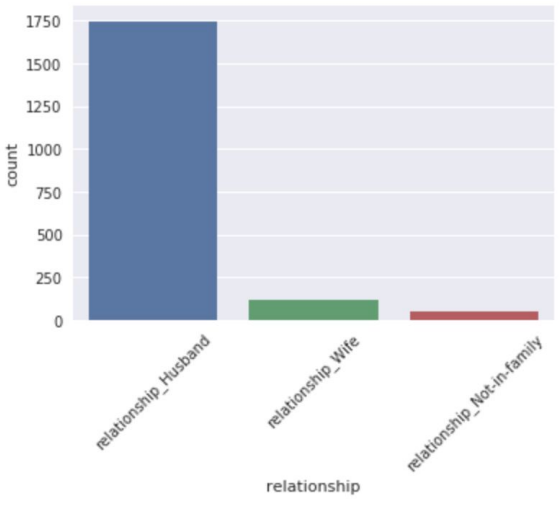


We also generate various summary inferences filtering out only section 1 individuals to target them specifically. This is done to identify the main attributes that result in high income. On the right side, you see the top working class of individuals in section 1. Most of them work in the private sector. Some individuals are also self employed and work for the federal government.



Occupations like professors or executive managerial positions were extremely high in section 1 compared to other occupations like sales, clerics etc. We also noticed that most of the individuals in section 1 were married husbands as shown in the right chart. As an NGO, we could use all these attributes and insights to target individuals and request appropriate donation amounts.





Future work

Through this project, we analysed the potential of using sophisticated classification algorithms to determine an individual's income. We also came up with a strategy of targeting an individual with an optimum donation request that could possibly lead to an increase in the funding status of an NGO.

This project could be scaled in multiple ways. Some of them are listed as follows:

- Implementing a recommender engine to match an individual to a donation request in a more granular fashion using previous transaction details and various similarity indices.
- Appending more data points and features to the existing model to improve model classification performance and result in accurate matching.
- Donation amounts provided by individuals could be incorporated into the model to improve and optimise donation requests, hence leading to higher funds for the NGO.

Conclusion

Through exploring the adult dataset found on the UC Irvine Machine Learning Repository, we gained interesting insights into different factors that determine a person's income. The dataset was not perfect, and using data exploration/visualizations, we were able to identify the problems with the dataset and address them accordingly. After cleaning and preprocessing the data, we were able to explore some additional functionality of Sagemaker, such as the linear learner model and using a Sklearn function as an endpoint, while also gaining additional exposure to the XGBoost model which was used in previous assignments.

While Sagemaker is a powerful tool, this project also exposed some of its drawbacks. Specifically, the lack of feature weights/importance extraction in the linear learner model was a huge drawback as well as its lack of interaction functionality. It was also quite surprising that Sagemaker does not come with a Random Forest model, and we spent a considerable amount of time configuring a Sklearn Random Forest model to be used in Sagemaker. In terms of debugging our code, Sagemaker is quite new and for this reason lacks the same volume of online discussion as other big data processing tools such as Spark. We anticipate that Sagemaker will address some of these drawbacks in the near future.

After a comparison of evaluation metrics calculated among the predictions of our models, we finalized our decision to use XGBoost as our final model. The insights gained from the prediction of the XGBoost model paint a useful picture of those who are more likely to donate to NGOs. We hope that the analysis we have provided can be used to support the funding of NGOs. In such challenging times, NGOs will play a huge role in addressing the challenges that the world currently faces, as well as challenges that loom on the horizon.