ML-1 Challenge 2: High Income Group Prediction

APRIL 10, 2022

MUHAMMAD AMIN GHIAS -25366

Table of Contents

ABSTRACT:

Getting to know income range of people is important. This project will predict with the given features whether the person lies in high income group or not.

INTRODUCTION

Earning money is important for survival of humans. Having high income is the goal and need of many. Knowing which people are in high income group is beneficial in many aspects, like surveys, marketing etc.

BUSINESS PROBLEM & MAIN OBJECTIVE

Finding the income range of a person is important in terms of sales team, so they can target their products for people with that income range. This project will predict based on the given 13 features whether person lies in high income range or not

DATA ACQUISITION AND WRANGLING

DATA SOURCE

For this project we will use online data the source is from kaggle

with url:" https://www.kaggle.com/competitions/ml-1-challenge-2-high-income-group-prediction/data"

The data set is inspired from a similar dataset posted earlier on Kaggle. However, this one has been changed significantly (through SMOTE) and altered and as such the distribution has been changed significantly.

2 different dataset are given, 1 train dataset and 1 test dataset is provided

The train dataset consists of 15 columns and 68378 rows

We have to predict whether a person belongs to a high-income group or not using the following 13 variables:

a) Age: Numeric

b) WorkClass: Categorical

c) Education Level: Categorical

d) Marital Status: Categorical

e) Occupation: Categorical

f) Gender: Categorical

g) Hours Per Week Working: Numeric

h) Native Country: Categorical

In addition, there are 5 masked variables (for which semantics/domain information is not provided):

i) X1: Numeric

j) X2: Numeric

k) X3: Categorical

l) X4: Numeric

m) X5: Numeric

DATA DESCRIPTION

A brief description of dataframe is as:

```
df.head()
```

0.2s

	row ID	Age	WorkClass	X1	Education Level
0	Row2	38.0	2	215646.0	1
1	Row3	53.0	2	234721.0	2
2	Row5	37.0	2	284582.0	3
3	Row7	52.0	1	209642.0	1
4	Row8	31.0	2	45781.0	3

```
df.shape
```

```
(68378, 15)
```

```
df.info()
✓ 0.1s

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 68378 entries, 0 to 68377
Data columns (total 15 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   row ID                  68378 non-null  object
 1   Age                     68378 non-null  float64
 2   WorkClass               68378 non-null  int64
 3   X1                      68378 non-null  float64
 4   Education Level         68378 non-null  int64
 5   X2                      68378 non-null  float64
 6   Marital Status          68378 non-null  int64
 7   Occupation              68378 non-null  int64
 8   X3                      68378 non-null  int64
 9   Gender                  68378 non-null  int64
 10  X4                      68378 non-null  float64
 11  X5                      68378 non-null  float64
 12  Hours Per Week Working  68378 non-null  float64
 13  Native Country          68378 non-null  int64
 14  High Income             68378 non-null  int64
```

The target feature is **high Income** category which we have to predict

DATA CLEANING & TRANSFORMATION

We will convert out categorical features which are as integers into string so we can apply onehot-encoding on them.

```
df1['WorkClass'] = df1['WorkClass'].astype('str')

df1['Education Level'] = df1['Education Level'].astype('str')

df1['Marital Status'] = df1['Marital Status'].astype('str')

df1['Occupation'] = df1['Occupation'].astype('str')

df1['Gender'] = df1['Gender'].astype('str')

df1['Native Country'] = df1['Native Country'].astype('str')


df1['X3'] = df1['X3'].astype('str')

# Deleted row ID from df1
df1.drop(['row ID'], axis=1, inplace=True)
✓ 0.6s
```

Using onehot encoding

```
df_onehot = pd.get_dummies(df1)
df_onehot.dtypes
```

✓ 0.3s

```
Age                    float64
X1                     float64
X2                     float64
X4                     float64
X5                     float64
                         ...
Native Country_5         uint8
Native Country_6         uint8
Native Country_7         uint8
```

In this way we form our X and y for training data and Xt for test data

Forming X and y of training data

```
dfX=df_onehot.copy()
dfX=dfX.drop(columns=['High Income'])
print(dfX.shape)
dfX.head()
X=dfX
X.head()
y=df_onehot['High Income']
y.head()
```

✓ 0.1s

```
(68378, 102)
```

After the onehot-encoding we get 102 features(columns).

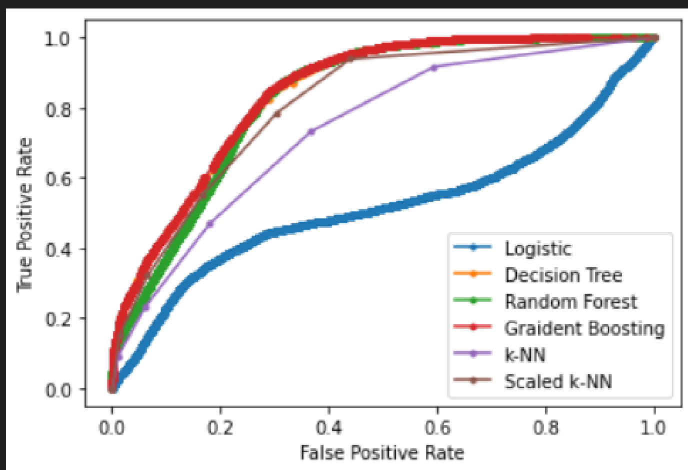
DATA ANALYSIS & USE OF MODELS:

Applying different models on our dataset

We use train test split method to on train data to test our different models and find their AUC score.

And get the following results

```
Logistic  :  0.5198236084149601
Decision Tree  :  0.836374728092627
Random Forest  :  0.8311080809430372
Graident Boosting  :  0.8453269699803782
k-NN  :  0.7383664304969388
Scaled k-NN  :  0.8090227020279421
```



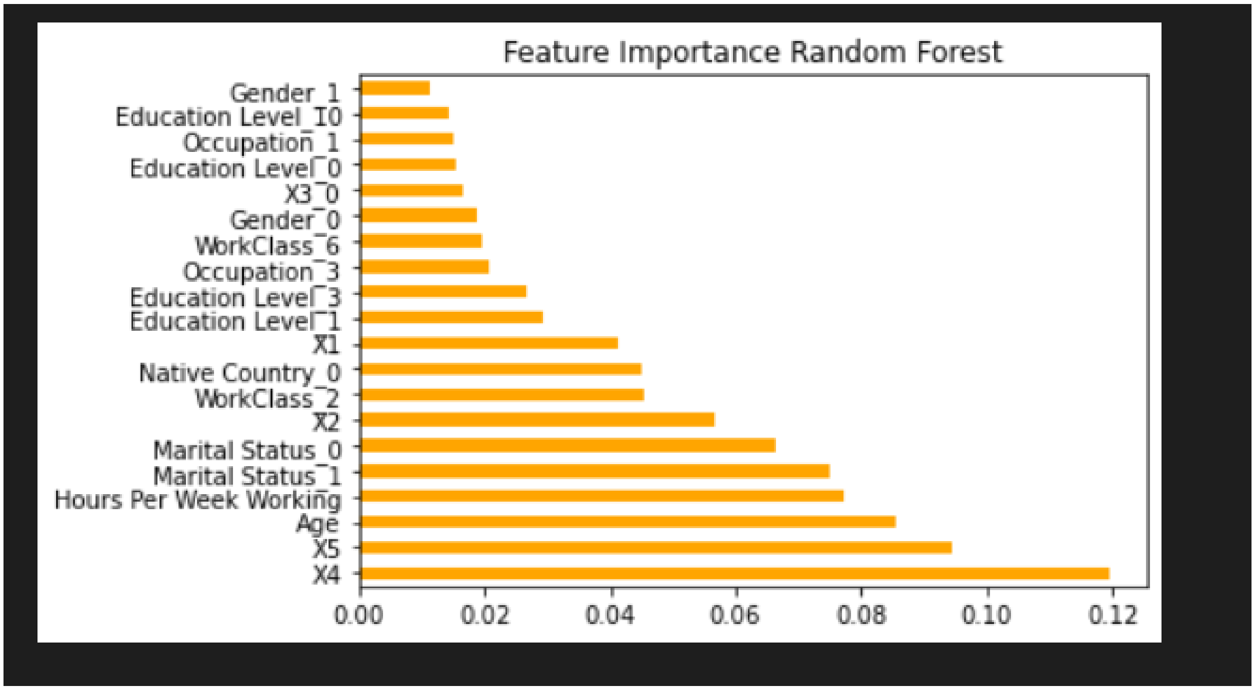
From the above results it shows that 3 models work better with higher AUC score

1. Decision Tree
2. Random Forrest Classifier
3. Gradient Boosting

The following models were applied on the complete train test data

1. Logistic Regressor
2. KNN-Classifer
3. Gaussian NB
4. Decision Tree
5. Random Forrest Classifier
6. Gradient Boosting

Also, we found Important features using Random Forrest Classifier



Before proceeding with detailed analysis, we performed one model once on the 3 datasets

1. Complete dataset
2. Filtered dataset
3. Dataset of top 70 features based on Random Forrest Classifier important features

We found and concluded that best results were obtained with completed training dataset for our particular dataset.

Hence further all analysis and models were tested on completed data.

PARAMETERS OPTIMIZATION AND SELECTION:

The 3 models Decision Tree, Random Forrest Classifier and Gradient Boosting were run with different parameters to find which parameters give best results

```
results2.sort_values(by=['AUC'],ascending=False)
```

	Name	depth	estimators	AUC
29	RandomForestClassifier	30	100	0.892559
33	RandomForestClassifier	34	100	0.892470
34	RandomForestClassifier	35	100	0.892142
43	RandomForestClassifier	44	100	0.892048
70	RandomForestClassifier	71	100	0.892047
...	...	...	...	...
4	RandomForestClassifier	5	100	0.832476
3	RandomForestClassifier	4	100	0.821505
2	RandomForestClassifier	3	100	0.808331
1	RandomForestClassifier	2	100	0.797752

Name	depth	estimators	AUC
RandomForestClassifier max_features=20	30	1000	0.893536
Scaled RandomForestClassifier max_features=20	30	800	0.893526
RandomForestClassifier max_features=20	30	1000	0.89335
RandomForestClassifier max_features=20	30	1000	0.893232
Scaled RandomForestClassifier max_features=20	30	2000	0.893173
RandomForestClassifier max_features=15	30	1000	0.893172
Scaled RandomForestClassifier max_features=20	30	1000	0.893148
Scaled RandomForestClassifier max_features=20	30	600	0.893144
RandomForestClassifier max_features=20	30	800	0.892876
Scaled RandomForestClassifier max_features=15	30	1000	0.892825
RandomForestClassifier max_features=30	30	1000	0.892737
RandomForestClassifier max_features=25	30	1000	0.892726

Similarly, the best parameters of decision Tree and Gradient boosting were found.

BEST MODELS & PARAMETERS

The parameters of the top 5 best models are as follows:

S.No	Name	AUC
1	GradientBoostingClassifier(max_depth=16,n_estimators=3000,max_features=20, verbose=2)	0.91161
2	GradientBoostingClassifier(max_depth=16,n_estimators=3200,max_features=30, verbose=2)	0.91141
3	GradientBoostingClassifier(max_depth=16,n_estimators=2700,verbose=2)	0.91133
4	GradientBoostingClassifier(max_depth=16,n_estimators=2500,verbose=2)	0.91111
5	GradientBoostingClassifier(max_depth=16,n_estimators=2000,verbose=2)	0.91083

Stacking Classifier

Next, we applied Stacking Classifier on 3 best models decision tree, Random Forrest Classifier and gradient boosting the results of AUC obtained was not an improvement.

Stacking was also done manually on the output results of our 4 best model results using the random Forrest Classifier and gradient boosting as out final predictor model. The results of these were not an improvement either

BEST RESULT:

In the end we used the approach of finding the average of the output predicted results of our top 5 best models.

This gave the best results with an AUC of **0.91229**

```
xtst.head()
```

	s1	s2	s3	s4	s5	average
0	4.969837e-01	3.011290e-02	2.143841e-02	8.879065e-01	5.957155e-01	4.064314e-01
1	4.160000e-10	4.170000e-10	1.560000e-09	2.120000e-09	2.640000e-09	1.430600e-09
2	4.990000e-09	3.450000e-09	8.670000e-09	9.720000e-09	1.470000e-08	8.306000e-09
3	3.140000e-10	9.920000e-11	1.180000e-09	5.190000e-11	6.530000e-10	4.596200e-10
4	6.974059e-01	1.265695e-01	8.530460e-01	9.810692e-01	5.837497e-01	6.483680e-01

```
Xto=pd.read_csv('xts.csv')
ro=Xto['row ID']
type(ro)
row=ro.values
prd=pd.DataFrame({'row ID' : row, 'High Income': xtst['average']}, index=None)
prd
prd.head()
prd.to_csv('o63.csv',index=False)
```

CONCLUSION

During our analysis the models which gave better results of AUC were the decision Tree, Random Forrest and Gradient Boosting.

Following major observations concluded:

- Filtering did not improve AUC Results
- Feature selection with random Forrest important features (top 70) did not increase AUC.
- Stacking method applied didn't give better results
- GridSearch took too much time to give the optimum parameters and could not be used efficiently.
- The best model which gave the top 5 results was **gradient boosting** with **max_depth=16, n_estimators=3000, max_features=20, and AUC** of **0.91161.**
- Applying mean on the outputs of best 5 resulting models gave the best result with AUC **0.91229**

SUGGESTIONS

Some suggestions for future work are:

- Using gridsearch and finding best parameters for all models may increase results.