|  |  |
| --- | --- |
| Compensation vs. Salary: Make the Difference Work for You - Workest  Income Prediction analysis  ML Techniques: Random Forest Classifiers and XG Boost | Ravi, Sneha |

**Background:**

In today's rapidly evolving workforce, understanding the factors that contribute to an individual's income is crucial for both policymakers and individuals seeking financial stability. To shed light on this important topic, we turn to the "Adult Income Dataset," a comprehensive collection of information on various socio-economic attributes and earnings of individuals.

The dataset comprises demographic features such as age, education level, race, sex, work class, occupation, and native country, along with continuous variables like capital gain, capital loss, and hours worked per week. The event of interest in this study is whether an individual earns more or less than $50,000 annually, making it a binary classification problem.

**A look into the Statistical Methods:**

**Random Forest Classifier:**

Random Forest uses multiple decision trees to make predictions using a method called Wisdom of Crowds. It takes random samples of both the data and the features considered to make predictions, which helps improve accuracy. However, understanding the reasons behind its predictions can be challenging because it is considered a black-box model, and it may sometimes give error because of overfitting the model.

**XG Boost:**

XGBoost is a widely used boosting software for making accurate predictions. It's computationally efficient and available for popular data science languages like R and Python. XGBoost has various parameters, including "subsample," controlling the fraction of sampled observations, and "eta", which prevents overfitting by controlling the weights' change during boosting. Adjusting these parameters can enhance the model's performance, and using subsample mimics the behavior of random forests.

**Data:**

**Salary:** Salary below or above 50K

**Age:** Age, continuous variable.  
**Workclass:** Type of Organization  
**Census\_weight:** Weight of census, continuous variable.  
**Education:** Education level  
**Marital-status:** Marital status

**Occupation:** Type of occupation

**Relationship:** Relationship status

**Sex:** Female, Male.  
**Capital-gain:** Capital gains from investments, continuous variable.  
**Capital-loss:** Capital losses from investments, continuous variable.

**Hours-per-week:** Number of working hours per week, continuous variable.  
**Native-country:** Native country

Snippet of the data

Fig 1.0

A screen shot of a computer

Description automatically generated

Count plot to show the distribution of salary. A significantly larger number of individuals earn below 50K than those earning above.

Fig 1.1

A graph of a distribution of salary

Description automatically generated

Visualize the distribution of education with salary. HS-grad: majority earn <=50K, Bachelors: majority earn >50K

Fig 1.2:

A graph of salary and salary

Description automatically generated

Count plot reveals the occupation-salary relationship. Adm-clerical: majority earn <=50K, Exec-manager: majority earn >50K.

Fig 1.3:

A graph of different colored lines

Description automatically generated

Visualize the distribution of age with salary. Age 22: majority earn <= 50K, Age 45: majority earn >50K.

Fig 1.4:

A graph of age with different colored bars

Description automatically generated

**Data Exploration and Prep:**

The initial data exploration involves checking the dataset's information to understand the data types and non-null counts. It also checks for any missing values and identifies any duplicate rows. To ensure data cleanliness, duplicate rows are removed, this reduced the number of rows from 31977 to 31955, and the resulting dataset is stored in a new Data Frame called 'adult'. The next step involves converting categorical variables into numeric representations. The 'education', 'salary', 'sex', 'race', 'workclass', and 'occupation' columns are encoded to transform them into ordinal or nominal values. This is essential for machine learning algorithms, as they often require numeric inputs.

Basic information about all the variables in the dataset. This includes details such as variable names, data types.

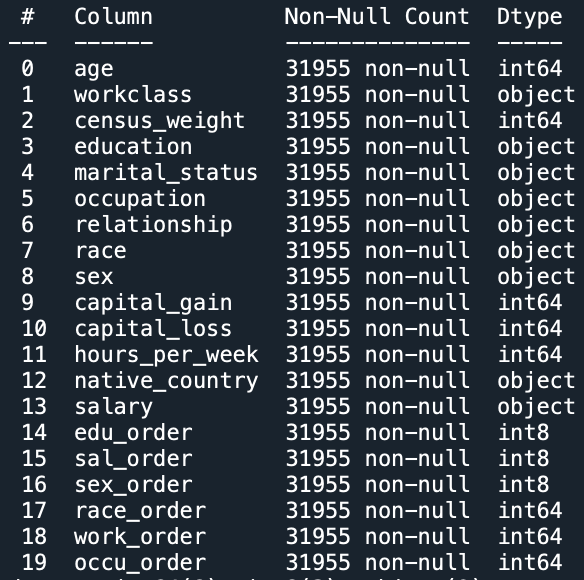
Fig 2.1:

A screen shot of a computer

Description automatically generated

After applying data filtering, eliminating duplicates, and converting categorical variables into numeric values, we obtain basic information about the dataset with the added numerical columns.

Fig 2.2:



The data is split into two sets: the training set and the test set. The split is done with an 80-20 ratio, meaning 80% of the data is allocated to the training set, while the remaining 20% is allocated to the test set. This division allows for model training on the training set and evaluation on the test set to assess its performance.

The training dataset has 25564 rows of data.

Fig 2.3:

A screenshot of a computer code

Description automatically generated

The testing dataset has 6391 rows of data.

Fig 2.4:

A screen shot of a computer

Description automatically generated

**Data Analysis:**

The model used is Random Forest, which combines many decision trees to make predictions. It's an ensemble learning method that trains each tree on random data and variables. The final prediction is based on all the trees' votes. Two models were created with different criteria: Entropy and Gini. They were trained on the data and evaluated for accuracy. The confusion matrix shows how well they predicted true positives and negatives. The goal is to compare their performance and find the best model for accurate predictions on new data.

**Random Forest Entropy:**

The confusion matrix for the entropy model provides essential insights into its performance. It indicates that there are 4730 true negatives (correctly predicted salary below 50K) and 731 true positives (correctly predicted salary above 50K). However, there are also 140 false positives (incorrectly predicted salary above 50K) and 790 false negatives (incorrectly predicted salary below 50K).

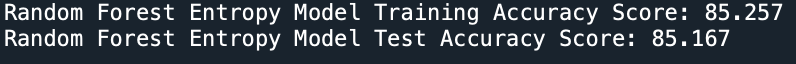
Fig 3.01:

A screenshot of a graph

Description automatically generated

With a training accuracy score of approximately 85.25% and a test accuracy score of about 85.16% , the model is performing consistently well on both the training and test datasets. Also, the close test score to the training score suggests that there is no overfitting, making the model more accurate and reliable in its predictions.

Fig 3.02:



ROC (Receiver Operating Characteristic) curve with recall and specificity, the x-axis represents the false positive rate (1 - specificity), and the y-axis represents the true positive rate (recall or sensitivity. When the curve leans towards the bottom right, it indicates that the model is effective at correctly identifying positive instances while also keeping the number of false positives relatively low.

Fig 3.03:

A graph of a forest roc curve

Description automatically generated

**Random Forest Gini:**

The Gini model's confusion matrix shows its performance in binary classification. It correctly predicted 741 true positives (correctly predicted salary above 50K) and 4715 true negatives (correctly predicted salary below 50K) but misclassified 780 as false negatives (incorrectly predicted salary below 50K) and 155 as false positives (incorrectly predicted salary above 50K). It demonstrates reasonable performance, but some misclassifications suggest room for improvement

Fig 3.04:

A graph with numbers and a number of negatives

Description automatically generated

The Gini model achieved a training accuracy score of 85.53% and a test accuracy score of 85.37%. This indicates that the model is performing consistently on both the training and test datasets. Also, the close test score to the training score suggests that there is no overfitting, making the model more accurate and reliable in its predictions.

Fig 3.05:



**Feature Importance:**

Evaluate the importance of features using the Random Forest Entropy Model. Features are ranked based on their impact on model accuracy. The process involves splitting the data into training and testing sets, and then training the model. Feature importance is assessed by comparing predictions before and after random permutation. The results show features sorted by importance scores, indicating their influence on model accuracy. This helps understand which factors are crucial for the model's performance.

Capital gain and education are identified as the top two most important factors influencing the model's accuracy. This indicates that these two variables have a significant impact on predicting the outcome. The least important factors are race, workclass and occupation. In other words someone with a higher capital gain and education would have a higher change of earning more than 50K.

Fig 3.06:

A graph of a bar graph

Description automatically generated

**XG Boost – Random Forest:**  
Relevant features (age, education, sex, work class, hours per week, occupation, capital gain, and capital loss) are selected as input (X) for the XGBoost classification model. The target variable (salary category) is extracted as the output (Y). The dataset is split into training and testing sets to train the model and evaluate its accuracy.

XG Boost random forest model analyses 8 columns after dropping race order because of the very low score from the feature importance.

Fig 3.07:

A screenshot of a computer screen

Description automatically generated

The model correctly predicted 4593 instances where the actual category was 0 (earning below 50K), and it correctly predicted 880 instances where the actual category was 1 (earning above 50K). However, it misclassified 683 instances as 1 when they were 0 (false positives), and 235 instances as 0 when they were 1 (false negatives).

Fig 3.08:

A graph with numbers and a number on it

Description automatically generated

The XGBoost model demonstrates good performance in predicting salary categories. During training, the model achieved an accuracy score of 87.73%, while on the testing dataset, it achieved an accuracy score of 85.64%. Although there is approximately 2% between the test and train accuracy, there is significant evidence of overfitting, therefore the model is accurate.

Fig 3.09:



The curve indicates that the model is effective at correctly identifying positive instances while also keeping the number of false positives relatively low. Hence the model is accurate.

Fig 3.1:

A red line with a green line

Description automatically generated

**Feature Importance:**

The process for calculating feature importance for XGBoost is identical to that used for entropy. In the XGBoost model, capital gain, sex, and education are the top three most important features for making predictions. They have the highest impact on the model's accuracy. On the other hand, workclass, hours\_per\_week, and occupation are the bottom three features, meaning they have the least influence on the model's predictions. In practical terms, it suggests that factors like work class, weekly working hours, and occupation may not play a significant role in determining someone's earning potential compared to other critical factors like capital gains, education, and gender.

Fig 3.2:

A graph of a bar graph

Description automatically generated

The XG Boost model performed better than the Random Forest model with entropy in terms of accuracy and prediction. The XG Boost model achieved higher accuracy scores for both training and testing, around 87.7% and 85.6%, respectively, while the Random Forest model scored around 85.4% for both. The XG Boost model also showed better precision and recall for both salary categories, indicating it made more accurate predictions for both high and low earners. As a result, the XG Boost model is considered the better one and chosen as the final model for any future implementation.

**Conclusion:**

In conclusion, we conducted a study using the XG Boost model and the entropy model to predict salaries based on various factors. The XG Boost model outperformed the entropy model with higher accuracy and better predictions for both salary categories. It identified important features like capital gain, sex, and education, which strongly influence salary predictions. Individuals with higher capital gains, being male, and having higher levels of education are more likely to earn higher salaries. On the other hand, the least important predictor variables are "workclass," "hours\_per\_week," and "occupation." This implies that these factors have relatively less influence on an individual's earnings. This study shows the potential of using machine learning algorithms like XG Boost for accurate salary predictions.

**Appendix:**

**Code used for the analysis:**

import os

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from pandas.api.types import CategoricalDtype

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import RandomForestClassifier

from sklearn.preprocessing import LabelEncoder

from sklearn.metrics import confusion\_matrix

from xgboost import XGBClassifier

from sklearn import metrics

from collections import defaultdict

from dmba import plotDecisionTree, textDecisionTree

import pydotplus

from graphviz import Digraph

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import confusion\_matrix, precision\_recall\_fscore\_support

from sklearn.metrics import roc\_curve, accuracy\_score, roc\_auc\_score

import dmba as dmba

from sklearn.metrics import confusion\_matrix, precision\_recall\_fscore\_support

from sklearn.metrics import roc\_curve, accuracy\_score, roc\_auc\_score

os.chdir(r'/Users/sneharavi/Desktop/Quantitaive\_Methods/Week\_7')

os.getcwd()

df = pd.read\_csv('adult.csv')

df.info()

df.isna().sum()

df.duplicated().sum()

sns.countplot(data = df, x='sex', hue ='salary', palette="Set2")

%matplotlib inline

sns.countplot(data = df, x= 'education', hue ='salary', palette="flare")

plt.title('Distribution of Education with Salary')

plt.xticks(rotation = 90)

sns.countplot(data = df, x= 'occupation', hue ='salary', palette="husl")

plt.title('Occupation vs Salary')

plt.xticks(rotation = 90)

%matplotlib inline

sns.countplot(data = df, x= 'salary', palette="Spectral")

plt.title('Distribution of Salary')

sns.histplot(data=df, x='age', hue= 'salary', palette="dark")

plt.title('Distribution of Age with Salary')

adult = df.drop\_duplicates()

adult.head()

adult.info()

adult.shape

adult.columns

adult['salary'].unique()

#df.to\_excel('adult.xlsx')

#-----------------------------------------------------------------------------

# Convert to ordinal and Nominal Variables

# Education: 0 = Doctorate, 15 = Preschool

adult.education.unique()

ordered\_education = [' Doctorate',' Prof-school',' Masters',' Bachelors', ' Assoc-voc' ,' Assoc-acdm',' Some-college',' HS-grad',' 12th' ,' 11th', ' 10th',' 9th',' 7th-8th', ' 5th-6th', ' 1st-4th',' Preschool']

adult['edu\_order'] = adult.education.astype(CategoricalDtype(categories=ordered\_education, ordered=True)).cat.codes

adult.edu\_order.value\_counts()

# Salary: 1 = >50K, 0 = <=50K

ordered\_salary = [' <=50K', ' >50K']

adult['sal\_order'] = adult.salary.astype(CategoricalDtype(categories=ordered\_salary, ordered=True)).cat.codes

adult.sal\_order.value\_counts()

adult.salary.value\_counts()

# Sex: 1 = Female, 0 = Male

adult.sex.unique()

ordered\_sex = [' Female', ' Male']

adult['sex\_order'] = adult.sex.astype(CategoricalDtype(categories=ordered\_sex, ordered=True)).cat.codes

adult.sex\_order.value\_counts()

# Race: 0 = Amer-Indian-Eskimo, 1 = Asian-Pac-Islander, 2 = Black, 3 = Other, 4 = White

race = LabelEncoder()

adult['race\_order'] = race.fit\_transform(adult['race'])

adult.head()

adult.race\_order.value\_counts()

adult.race.value\_counts()

# 0 = Federal-gov , 1 = Local-gov, 2 = Never-worked, 3 = Private, 4 = Self-emp-inc, 5 = Self-emp-not-inc, 6 = State-gov, 7 = Without-pay, 8 = Unknown

work = LabelEncoder()

adult['work\_order'] = work.fit\_transform(adult['workclass'])

adult.work\_order.value\_counts()

adult.workclass.value\_counts()

# Occupation

occupation = LabelEncoder()

adult['occu\_order'] = occupation.fit\_transform(adult['occupation'])

adult.occu\_order.value\_counts()

#-----------------------------------------------------------------------------

# Set the Predictor and Outcome Variable

predictors = adult[['age','edu\_order','race\_order','sex\_order','work\_order','hours\_per\_week', 'occu\_order', 'capital\_gain', 'capital\_loss']]

outcome = adult['sal\_order']

outcome

# Split the Data

x\_train, x\_test, y\_train, y\_test = train\_test\_split(predictors, outcome, test\_size=0.20,random\_state=42)

x\_train.info()

y\_train.info()

x\_train.shape

x\_test.info()

#-----------------------------------------------------------------------------

# Model: RandomForest

# Criterion: Entropy

model\_entrop = RandomForestClassifier(n\_estimators=35, criterion='entropy', max\_depth=10, oob\_score=True)

model\_entrop.fit(x\_train, y\_train)

print('Random Forest Entropy Model Training Accuracy Score:' ,round(model\_entrop.score(x\_train, y\_train)\*100,3))

print('Random Forest Entropy Model Test Accuracy Score:', round(model\_entrop.score(x\_test, y\_test)\*100,3))

# RandomForest: Accuracy

pred = model\_entrop.predict(x\_test)

pred

cm = confusion\_matrix(y\_test, pred)

cm

print(cm[0,0])

confusion = dmba.classificationSummary(y\_test, pred, class\_names=model\_entrop.classes\_)

labels = ([['True Negative: %s'%cm[0,0], 'False Positive: %s'%cm[0,1]],['False Negative: %s'%cm[1,0], 'True Positive: %s'%cm[1,1]]])

fig, ax = plt.subplots()

sns.color\_palette("rocket")

sns.heatmap(cm, annot= labels, fmt = '', cmap="YlGnBu")

plt.title('Confusion Matrix: Random Forest Classifier (Entropy)')

# Roc curve

fpr, tpr, thresholds = roc\_curve(y\_test, model\_entrop.predict\_proba(x\_test)[:, 0])

roc\_entrop = pd.DataFrame({'recall': tpr, 'specificity': 1 - fpr})

plt.title('Random Forest ROC Curve')

plt.plot(fpr, tpr, color = 'Blue')

plt.plot([0, 1], [0, 1], 'r--')

plt.xlim([0, 1])

plt.ylim([0, 1])

plt.ylabel('Recall')

plt.xlabel('Specificity')

plt.show()

#-----------------------------------------------------------------------------

# Criterion: Gini

model\_gini = RandomForestClassifier(n\_estimators=35, criterion='gini', max\_depth=10, oob\_score=True)

model\_gini.fit(x\_train, y\_train)

model\_gini.score(x\_train, y\_train)

model\_gini.score(x\_test, y\_test)

print('Random Forest Gini Model Training Accuracy Score:' ,round(model\_gini.score(x\_train, y\_train)\*100,3))

print('Random Forest Gini Model Test Accuracy Score:', round(model\_gini.score(x\_test, y\_test)\*100,3))

# RandomForest: Accuracy

pred\_gini = model\_gini.predict(x\_test)

pred\_gini

cm\_gini = confusion\_matrix(y\_test, pred\_gini)

cm\_gini

dmba.classificationSummary(y\_test, pred\_gini, class\_names=model\_gini.classes\_)

confusion\_gini = dmba.classificationSummary(y\_test, pred\_gini, class\_names=model\_gini.classes\_)

labels = ([['True Negative: %s'%cm\_gini[0,0], 'False Positive: %s'%cm\_gini[0,1]],['False Negative: %s'%cm\_gini[1,0], 'True Positive: %s'%cm\_gini[1,1]]])

fig, ax = plt.subplots()

sns.heatmap(cm\_gini, annot= labels, fmt = '', cmap= 'Blues')

plt.title('Confusion Matrix: Random Forest Classifier (Gini)')

#-----------------------------------------------------------------------------

# Feature Importance Entropy

scores = defaultdict(list)

for \_ in range(3):

x\_train, x\_test, y\_train, y\_test = train\_test\_split(predictors, outcome, test\_size=0.20,random\_state=42)

model\_entrop.fit(x\_train, y\_train)

acc = metrics.accuracy\_score(y\_test, model\_entrop.predict(x\_test))

for column in predictors.columns:

X\_t = x\_test.copy()

X\_t[column] = np.random.permutation(X\_t[column].values)

shuff\_acc = metrics.accuracy\_score(y\_test, model\_entrop.predict(X\_t))

scores[column].append((acc-shuff\_acc)/acc)

print('Features sorted by their score:')

print(sorted([(round(np.mean(score), 4), feat) for feat, score in scores.items()], reverse=True))

importances = model\_entrop.feature\_importances\_

importances

df1 = pd.DataFrame({'feature': predictors.columns,'Accuracy decrease': [np.mean(scores[column]) for column in predictors.columns],

'Gini decrease': model\_entrop.feature\_importances\_,

'Entropy decrease': model\_gini.feature\_importances\_,})

df1 = df1.sort\_values('Accuracy decrease')

fig, axes = plt.subplots(ncols=2, figsize=(8, 5))

ax = df1.plot(kind='barh', x='feature', y='Accuracy decrease',legend=False, ax=axes[0], color='pink')

ax.set\_ylabel('')

ax = df1.plot(kind='barh', x='feature', y='Gini decrease', legend=False, ax=axes[1], color = 'pink')

ax.set\_ylabel('')

ax.get\_yaxis().set\_visible(False)

plt.tight\_layout()

plt.show()

#-----------------------------------------------------------------------------

# XGBoost Classifier

X = adult[['age','edu\_order','sex\_order','work\_order','hours\_per\_week', 'occu\_order', 'capital\_gain', 'capital\_loss']]

Y = adult['sal\_order']

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, test\_size = 0.2)

X\_train.info()

Y\_train.info()

xgb = XGBClassifier(objective='binary:logistic', use\_label\_encoder=False, subsample = 1, eval\_metric = 'error' )

xgb.fit(X\_train, Y\_train)

print('XG Boost Model Training Accuracy Score:',round((xgb.score(X\_train, Y\_train))\*100,3))

print('XG Boost Model Testing Accuracy Score:',round((xgb.score(X\_test, Y\_test))\*100,3))

xgb\_predict = xgb.predict(X\_test)

xgb\_predict

cm\_xgb = confusion\_matrix(Y\_test, xgb\_predict)

cm\_xgb

confusion\_xg = dmba.classificationSummary(Y\_test, xgb\_predict, class\_names=xgb.classes\_)

labels = ([['True Negative: %s'%cm\_xgb[0,0], 'False Positive: %s'%cm\_xgb[0,1]],['False Negative: %s'%cm\_xgb[1,0], 'True Positive: %s'%cm\_xgb[1,1]]])

fig, ax = plt.subplots()

sns.heatmap(cm, annot= labels, fmt = '', cmap="BuPu")

plt.title('Confusion Matrix: XGB Classifier')

fpr, tpr, thresholds = roc\_curve(Y\_test, xgb.predict\_proba(X\_test)[:, 0])

roc\_xgb = pd.DataFrame({'recall': tpr, 'specificity': 1 - fpr})

plt.title('XGBoost ROC Curve')

plt.plot(fpr, tpr, color = 'green')

plt.plot([0, 1], [0, 1], 'r--')

plt.xlim([0, 1])

plt.ylim([0, 1])

plt.ylabel('Recall')

plt.xlabel('Specificity')

plt.show()

#-----------------------------------------------------------------------------

# Feature Importance XG Boost

importances = xgb.feature\_importances\_

df\_importance = pd.DataFrame({'Feature': X.columns, 'Importance': importances})

df\_importance = df\_importance.sort\_values(by='Importance', ascending=True)

plt.figure(figsize=(8, 5))

plt.barh(df\_importance['Feature'], df\_importance['Importance'], color='lightblue')

plt.xlabel('Importance')

plt.ylabel('Feature')

plt.title('XGBoost Model Feature Importance')

plt.show()

#-----------------------------------------------------------------------------

#Model Comparison

models\_df = pd.DataFrame()

models\_df['Observations\_entrop'] = ['Above 50K' if x == 1 else 'Below 50K' for x in y\_test]

models\_df['entropy\_prediction'] = ['Above 50K' if sal\_order == 1 else 'Below 50K' for sal\_order in model\_entrop.predict(x\_test)]

models\_df['Observations\_XGB'] = ['Above 50K' if x == 1 else 'Below 50K' for x in Y\_test]

models\_df['xgb\_prediction'] = ['Above 50K' if sal\_order == 1 else 'Below 50K' for sal\_order in xgb\_predict]

models\_df['prob\_entropy'] = model\_entrop.predict\_proba(x\_test)[:, 0]

models\_df['prob\_xgb'] = xgb.predict\_proba(X\_test)[:, 0]

print(models\_df.head(10))

#models\_df.to\_excel('prediction.xlsx')