Census Income Project

Introduction :

In this project, we are going to predict whether a person's income is above 50k or below 50k using various features like age, education, and occupation. The dataset we are going to use is the census income dataset which contains about 32560 rows and 15 features .

Defining the problem statement :

The data contains anonymous information such as age, occupation, education, working class, etc. *The goal is to train a binary classifier to predict the income which has two possible values '>50K' and '<50K'.* There are 48842 instances and 14 attributes in the dataset. The data contains a good blend of categorical, numerical and missing values.

The dataset contains the labels which we have to predict which is the dependent feature 'Income level'. This feature is discrete consisting of two categories income less than 50k and more than 50k. So the problem we have is a **Supervised Binary Classification** type.

Data Pre-processing Done:

The purpose of preprocessing is to convert raw data into a form that fits machine learning. Structured and clean data allows a data scientist to get more precise results from an applied machine learning model. The technique includes data formatting, cleaning, and sampling.

- **Hardware and Software Requirements and Tools Used**

- *Tools: Python 3.8.5, Jupyter Notebook, Numpy, Pandas, Matplotlib, Seaborn, Scikit-learn, Scipy*
- *Techniques: LogisticRegression, GaussianNB, SVC, RandomForestClassifier, AdaBoostClassifier, DecisionTreeClassifier, LinearSVC.*
- *Hardware: I3 processor, 4GB RAM*

- **Model/s Development and Evaluation :**

I have used the below models for classification:

1. LogisticRegression
2. SVC
3. RandomForestClassifier
4. AdaBoostClassifier
5. DecisionTreeClassifier
6. *GaussianNB*
7. *LinearSVC*
8. GradientBoostingClassifier
9. MultinomialNB
10. KNeighborsClassifier

- Identification of possible problem-solving approaches (methods)

- Read the data (from csv)
- Identify the dependent and independent variables.
- Check if the data has missing values or the data is categorical or not.
- If yes, apply basic data preprocessing operations to bring the data in a go to go format.
- Now split the data into the groups of training and testing for the respective purpose.
- After splitting data, fit it to a most suitable model. (How to find a suitable model is answered below)
- Validate the model. If satisfactory, then go with it, else tune the parameters and keep testing. In a few cases, you can also try different algorithms for the same problem to understand the difference between the accuracies.
- From step 7 one can also learn about accuracy paradox.
- Visualize the data.

- **Testing of Identified Approaches (Algorithms)**

1. Naive Bayes
2. Cross validation
3. Confusion matrix
4. Accuracy score
5. Classification report

- **Run and Evaluate selected models**

I have used the below models for classification:

Logistic regression

```
]: lg=LogisticRegression()
   lg.fit(x_train,y_train)
   pred=lg.predict(x_test)
   lg_accu=accuracy_score(y_test,pred)
   print(accuracy_score(y_test,pred))
   print(confusion_matrix(y_test,pred))
   print(classification_report(y_test,pred))
```

```
0.7849232201023731
[[7668  443]
 [1868  766]]
              precision    recall  f1-score   support

           0       0.80      0.95      0.87      8111
           1       0.63      0.29      0.40      2634

    accuracy                           0.78     10745
   macro avg       0.72      0.62      0.63     10745
weighted avg       0.76      0.78      0.75     10745
```

KNeighborsClassifier

```
from sklearn.neighbors import KNeighborsClassifier
knn=KNeighborsClassifier()
knn.fit(x_train,y_train)
knn_pred=knn.predict(x_test)
knn_accu=accuracy_score(y_test,pred)

print(accuracy_score(y_test,pred))
print(confusion_matrix(y_test,pred))
print(classification_report(y_test,pred))
```

```
0.7849232201023731
[[7668  443]
 [1868  766]]
              precision    recall  f1-score   support

           0       0.80      0.95      0.87      8111
           1       0.63      0.29      0.40      2634

    accuracy                           0.78     10745
   macro avg       0.72      0.62      0.63     10745
weighted avg       0.76      0.78      0.75     10745
```

RandomForestClassifier

```
rf=RandomForestClassifier()
rf.fit(x_train,y_train)
pred=rf.predict(x_test)
rf_accu=accuracy_score(y_test,pred)
print(accuracy_score(y_test,pred))
print(confusion_matrix(y_test,pred))
print(classification_report(y_test,pred))
```

```
0.8538855281526291
[[7520  591]
 [ 979 1655]]
              precision    recall  f1-score   support

           0       0.88      0.93      0.91      8111
           1       0.74      0.63      0.68      2634

    accuracy                           0.85     10745
   macro avg       0.81      0.78      0.79     10745
weighted avg       0.85      0.85      0.85     10745
```

SVC

```
svc=SVC()
svc.fit(x_train,y_train)
pred=svc.predict(x_test)
svc_accu=accuracy_score(y_test,pred)
print(accuracy_score(y_test,pred))
print(confusion_matrix(y_test,pred))
print(classification_report(y_test,pred))
```

```
0.7879944160074454
[[8101   10]
 [2268  366]]
              precision    recall  f1-score   support

           0       0.78      1.00      0.88      8111
           1       0.97      0.14      0.24      2634

    accuracy                           0.79     10745
   macro avg       0.88      0.57      0.56     10745
weighted avg       0.83      0.79      0.72     10745
```

AdaBoostClassifier

```
ad=AdaBoostClassifier()
ad.fit(x_train,y_train)
pred=ad.predict(x_test)
ad_accu=accuracy_score(y_test,pred)
print(accuracy_score(y_test,pred))
print(confusion_matrix(y_test,pred))
print(classification_report(y_test,pred))
```

```
0.8577943229409027
[[7596  515]
 [1013 1621]]
              precision    recall  f1-score   support

           0       0.88      0.94      0.91      8111
           1       0.76      0.62      0.68      2634

    accuracy                           0.86     10745
   macro avg       0.82      0.78      0.79     10745
weighted avg       0.85      0.86      0.85     10745
```

GaussianNB

```
gnb=GaussianNB()
gnb.fit(x_train,y_train)
pred=gnb.predict(x_test)
gnb_accu=accuracy_score(y_test,pred)
print(accuracy_score(y_test,pred))
print(confusion_matrix(y_test,pred))
print(classification_report(y_test,pred))
```

```
0.7892042810609586
[[7665  446]
 [1819  815]]
              precision    recall  f1-score   support

           0       0.81      0.95      0.87      8111
           1       0.65      0.31      0.42      2634

    accuracy                           0.79     10745
   macro avg       0.73      0.63      0.64     10745
weighted avg       0.77      0.79      0.76     10745
```

MultinomialNB

```
from sklearn.naive_bayes import MultinomialNB
mnb=MultinomialNB()
mnb.fit(x_train,y_train)
pred=mnb.predict(x_test)
mnb_accu=accuracy_score(y_test,pred)
print(accuracy_score(y_test,pred))
print(confusion_matrix(y_test,pred))
print(classification_report(y_test,pred))
```

```
0.7779432294090275
[[7743  368]
 [2018  616]]
              precision    recall  f1-score   support

           0       0.79      0.95      0.87      8111
           1       0.63      0.23      0.34      2634

    accuracy                           0.78     10745
   macro avg       0.71      0.59      0.60     10745
weighted avg       0.75      0.78      0.74     10745
```

LinearSVC

```
LSVC=LinearSVC()
LSVC.fit(x_train,y_train)
pred=LSVC.predict(x_test)
LSVC_accu=accuracy_score(y_test,pred)
print(accuracy_score(y_test,pred))
print(confusion_matrix(y_test,pred))
print(classification_report(y_test,pred))
```

```
0.76379711493718
[[8106    5]
 [2533  101]]
              precision    recall  f1-score   support

           0       0.76      1.00      0.86      8111
           1       0.95      0.04      0.07      2634

    accuracy                           0.76     10745
   macro avg       0.86      0.52      0.47     10745
weighted avg       0.81      0.76      0.67     10745
```

DecisionTreeClassifier

```
dt=DecisionTreeClassifier()
dt.fit(x_train,y_train)
pred=dt.predict(x_test)
dt_accu=accuracy_score(y_test,pred)

print(accuracy_score(y_test,pred))
print(confusion_matrix(y_test,pred))
print(classification_report(y_test,pred))
```

```
0.8078175895765473
[[7021 1090]
 [ 975 1659]]
              precision    recall  f1-score   support

           0       0.88      0.87      0.87      8111
           1       0.60      0.63      0.62      2634

    accuracy                           0.81     10745
   macro avg       0.74      0.75      0.74     10745
weighted avg       0.81      0.81      0.81     10745
```

GradientBoostingClassifier

```
gbc=GradientBoostingClassifier()
gbc.fit(x_train,y_train)
pred=gbc.predict(x_test)
gbc_accu=accuracy_score(y_test,pred)

print(accuracy_score(y_test,pred))
print(confusion_matrix(y_test,pred))
print(classification_report(y_test,pred))
```

```
0.8586319218241042
[[7664  447]
 [1072 1562]]
              precision    recall  f1-score   support

           0       0.88      0.94      0.91      8111
           1       0.78      0.59      0.67      2634

    accuracy                           0.86     10745
   macro avg       0.83      0.77      0.79     10745
weighted avg       0.85      0.86      0.85     10745
```

- Conclusion:

```
conclusion
```

	0	1	2	3	4	5	6	7	8	9	...	10735	10736	10737	10738	10739	10740	10741	10742	10743	10744
predicted	0	0	0	0	0	0	1	0	0	0	...	1	0	0	1	1	0	1	0	0	0
original	0	0	0	0	0	0	1	0	0	0	...	0	0	0	1	1	0	1	0	0	0

2 rows × 10745 columns

GradientBoostingClassifier is accuracy score ,precision ,recall ,CV and f1-score is good than other models,Hence GradientBoostingClassifier is performing is good. GradientBoostingClassifier has accuracy score is 85% .

GradientBoostingClassifier is best model for this dataset.