



iJRASET

International Journal For Research in
Applied Science and Engineering Technology



INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 7 Issue: XII Month of publication: December 2019

DOI: <http://doi.org/10.22214/ijraset.2019.12090>

www.ijraset.com

Call: ☎ 08813907089

E-mail ID: ijraset@gmail.com

Predict the Level of Income using Random Forest Classifier

Mr. Tejas Phase¹, Dr. Prof. Suhas Patil²

^{1,2}Department of Electronics, K. B. P. College of Enggi., Shivaji University Kolhapur, Maharashtra, India.

Abstract: Many social programs have a hard time making sure the right people receive the enough financial aid. It's tricky when a program focuses on the poorest segment of the population. This segment of population can't provide the necessary income and expense records to prove that they qualify to receive any financial aid under particular Govt. scheme. In this paper we predict someone's "Income Level" using the "Random Forest Classifier" based on the various attributes or features.

Keywords: Random Forest Classifier, PCA, Cross-Validation, Classification Report, Accuracy Score, Precision Score, Exploratory Data Analysis, Data Cleaning, NaN (Not a Number).

I. INTRODUCTION

In Latin America, a popular method called Proxy Means Test (PMT) uses an algorithm to verify income qualification. With PMT, agencies use a model that considers a family's observable household attributes like the material of their walls and ceiling or the assets found in their homes to classify them and predict their level of need or income. While this method is an improvement, accuracy remains a problem as the region's population grows and poverty declines. So, in this paper we proposed a system using "Random Forest Classifier" that can predict the "Income Level" of a particular family based on various household attributes of that family. But before inference of the "Income Level" we process our training and testing data. In this processing we are going to perform "Exploratory Data Analysis" in which we perform "Feature Engineering" which include the dealing with missing values that present in the training as well as in testing dataset. Then we perform "correlation analysis" between different feature columns and based on that select only particular columns that affects most in the task of prediction. Then we perform some "Data Cleaning" operations to get rid of "Categorical Columns" and lastly build our inference model.

II. EXPLORATORY DATA ANALYSIS

Here, first we load our dataset from our local directory and converts it into panda's dataframe format. After loading and conversion into dataframe format our data looks like below:

```
In [4]: #Review the dataset:
income_train_df.head(7)

Out[4]:
```

	Id	v2a1	hacdor	rooms	hacapo	v14a	refrig	v18q	v18q1	r4h1	...	SQBescolari	SQBage	SQBhogar_total	SQBbedjefe	SQBhogar_nin	S
0	ID_279628684	190000.0	0	3	0	1	1	0	NaN	0	...	100	1849	1	100	0	
1	ID_729eb3ddd	135000.0	0	4	0	1	1	1	1.0	0	...	144	4489	1	144	0	
2	ID_68de51c94	NaN	0	8	0	1	1	0	NaN	0	...	121	8464	1	0	0	
3	ID_d671db89c	180000.0	0	5	0	1	1	1	1.0	0	...	81	289	16	121	4	
4	ID_d56d9f5f5	180000.0	0	5	0	1	1	1	1.0	0	...	121	1369	16	121	4	
5	ID_ec05b1a7b	180000.0	0	5	0	1	1	1	1.0	0	...	121	1444	16	121	4	
6	ID_e9e0c1100	180000.0	0	5	0	1	1	1	1.0	0	...	4	64	16	121	4	

```
7 rows x 143 columns

In [5]: #View information about dataset:
income_train_df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9557 entries, 0 to 9556
Columns: 143 entries, Id to Target
dtypes: float64(8), int64(130), object(5)
memory usage: 10.4+ MB
```

Fig. 1. A High-level view of the training dataset

We are viewing first 7 records of the dataset using "income_train_df.head(7)" method and the information of the whole dataset using "income_train_df.info()" method. And by analysing the information about the dataset we can state that there are 9557 data records with 143 feature columns present in the dataset.

A. Treatment for Missing Values

We can check the presence of “missing values” in a particular column by calling “isnull()” command on that particular column like “income_train_df[‘v2a1’].isnull().sum()” whose output is: 6860. We can see that this column has huge number of missing values (around 6860). Ideally we drop column in this situation. But will decide whether to drop it or not based on its correlation with output variable.

We generally use “heat-maps” to see correlation of a particular variable with other. In our case we check the correlation of every feature column with “Target” variable to see how much that feature column useful during inference. This can be perform like following:

```
sns.heatmap(income_train_df[['v2a1','Target']].corr(),annot=True)
plt.show()
```

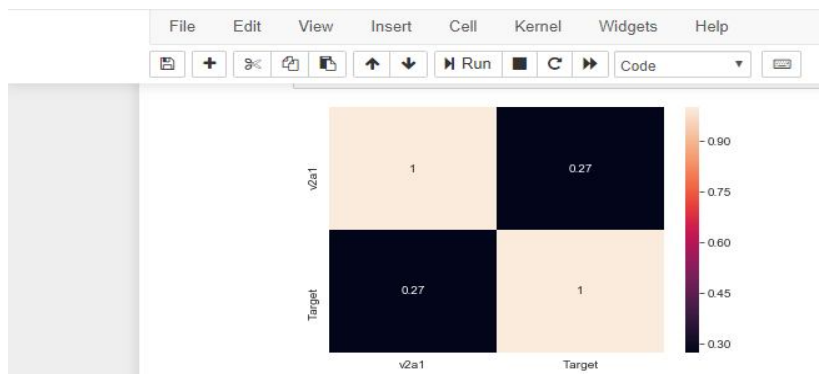


Fig. 2. Correlation Analysis with Heat-maps

By observing the above plot we can see that the correlation of feature column “v2a1” with output or “Target” variable is Positive but not much strong (0.27 in this case). In this case we use “Imputer class” to treat the missing values. So we going to impute values based on the number of rooms in the house. As column v2a1 is specifying monthly rent payment and by intuition we know that rent payment is depend on size of the house and size of the house is based on number of rooms. After performing imputation and other operations we check again the count of missing values in that column using the same command:

```
income_train_df[‘v2a1’].isnull().sum()
```

And this time we get the count: 2697 which is fairly low compared to previous.

Like-wise we perform this treatment for all the feature columns in the dataset to get rid of missing values.

B. Treatment for Categorical Columns

Here we convert categorical columns into numerical format and for that purpose we use numpy’s “where” method. This can be implemented as below:

Here we are showing this conversion only for one categorical column named “dependency”:

```
income_train_df[‘dependency’] = np.where(income_train_df[‘dependency’]==‘yes’,1,income_train_df[‘dependency’])
```

```
income_train_df[‘dependency’] = np.where(income_train_df[‘dependency’]==‘no’,0,income_train_df[‘dependency’])
```

In the above code we replace text “yes” with value “1” and “no” with value “0” and this conversion is performed in place.

Likewise we treat all the categorical columns in our training dataset and view the information about the dataset again:

```
income_train_df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9552 entries, 0 to 9551
Columns: 141 entries, v2a1 to Target
dtypes: float64(11), int64(130)
memory usage: 10.3 MB
```

Fig. 3. Dataset information after columns treatment

By comparing the first view of the information about the dataset and the current view we can clearly state that there are now only 2 types of datatypes present in the dataset which are float64 with 11 columns and int64 with 130 columns.

C. Separate feature and Target Column

For building the inference model we need to separate features from the Target column. And this can be done like following:

Separate feature columns and store it in variable X:

`X = income_train_df.iloc[:, :140].values.....All the rows and columns upto 140(not including)`

Separate Target column and store it in variable y:

`y = income_train_df.iloc[:, 140].values.....All the rows and only column 140`

D. Visualization of Target Column:

As we know, after separating Target column from other feature columns and store it in variable “y” which is by default numpy array and we can visualize this like following:

`y[:90]` ---We are viewing first 90 record's response:

```
In [75]: y[:90]
Out[75]: array([4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4,
4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4,
4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4,
4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4,
4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4,
4, 4], dtype=int64)

In [76]: y.shape
Out[76]: (9552,)
```

Fig. 4. Target Column

III.MODEL BUILDING

We perform this task using 2 approaches which include:

- 1) Building model without Dimensionality Reduction of the dataset.
- 2) Building model with Dimensionality Reduction dataset.

Here we deal with “Dimensionality Reduction” task using Principal Component Analysis. But for the first approach we don't go with Dimensionality Reduction task.

A. Building Model Without Dimensionality Reduction of the Dataset

- 1) *Split the Dataset:* We split the training data into 70% and 30% ratio. This means 70% of the whole dataset use for training purpose and rest 30% use for testing purpose. And here we set “random_state=22” to select the same samples again & again.

`X_train,X_test,y_train,y_test = train_test_split(X_train_norm,y,test_size=.3,random_state=22)`

After splitting the data, we are viewing the shape of the each respectively:

```
In [78]: print('X_train:',X_train.shape)
print('X_test:',X_test.shape)
print('y_train:',y_train.shape)
print('y_test:',y_test.shape)

X_train: (6686, 140)
X_test: (2866, 140)
y_train: (6686,)
y_test: (2866,)
```

Fig. 5 Shape of the Training and Testing dataset

- 2) *Classifier Instance Creation:* Then the next step is the creation of an instance of the “Random Forest” classifier:

`random_forrest_gini = RandomForestClassifier()`

`random_forrest_gini`

```
In [79]: #Create an Instance of the Random Forrest Classifier with default 'criterion=gini':
random_forrest_gini = RandomForestClassifier()
random_forrest_gini

Out[79]: RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
max_depth=None, max_features='auto', max_leaf_nodes=None,
min_impurity_decrease=0.0, min_impurity_split=None,
min_samples_leaf=1, min_samples_split=2,
min_weight_fraction_leaf=0.0, n_estimators='warn', n_jobs=None,
oob_score=False, random_state=None, verbose=0,
warm_start=False)
```

Fig. 6 Parameters of the Random Forest Classifier

- 3) *Fit the Dataset:* Fit the data into the model to make inferences: `random_forrest_gini.fit(X_train,y_train)`
- 4) *Make the predictions:* `y_predict = random_forrest_gini.predict(X_test)`
- 5) *Check Prediction Accuracy:* We can check model's prediction accuracy using different accuracy matrices:

a) Accuracy Score

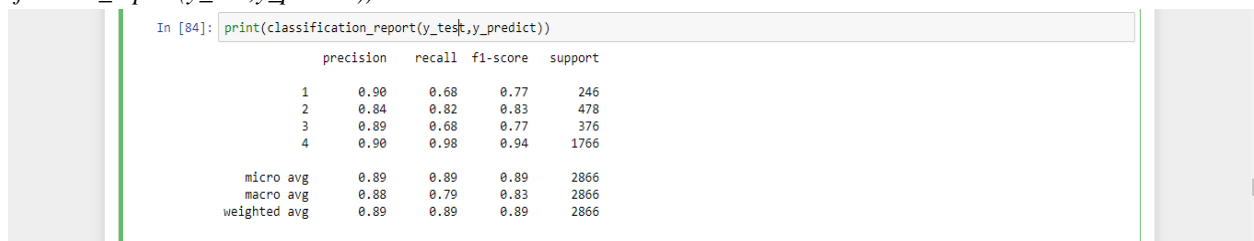
```
accuracy_with_gini = accuracy_score(y_test,y_predict)
print("Accuracy with criterion gini:",accuracy_with_gini)
```

0.890788554780182

We can see that our model accuracy is : 0.890788554780182

b) Classification Report

```
print(classification_report(y_test,y_predict))
```



```
In [84]: print(classification_report(y_test,y_predict))
```

	precision	recall	f1-score	support
1	0.90	0.68	0.77	246
2	0.84	0.82	0.83	478
3	0.89	0.68	0.77	376
4	0.90	0.98	0.94	1766
micro avg	0.89	0.89	0.89	2866
macro avg	0.88	0.79	0.83	2866
weighted avg	0.89	0.89	0.89	2866

Fig. 7. Model Classification Report

We can test our model for accuracy by changing different its parameters. In our case, we can change parameter “criterion” from “gini” to “entropy” and observe the model accuracy.

B. Building Model with Dimensionality Reduction Dataset

For this purpose we use “Principal Component Analysis”. This process include importing PCA module from sklearn then creation of an instance and fitting the original dataframe. After performing all these above steps we need to split the data into feature and target column like previous and after separating the data we feed it to the model. After building the model we check again its accuracy and this time we get:

Accuracy using PCA (gini): 0.7271458478715981

Accuracy using PCA (entropy): 0.7274947662247034

Which is low compared to the previous version of the model.

IV. CONCLUSIONS

Here we used “Random Forest” classifier to predict “Income Level” based on various attributes. We tested our Random Forest model on two types of datasets which are without dimensionality reduced and dimensionality reduced respectively. Again we observed different accuracy matrices by tuning parameters of the model which are “gini” and “entropy” respectively. We observed that we got higher accuracy on the dataset which is not dimensionality reduced than the dataset which is dimensionality reduced. Still there needs a room for improvement as higher accuracy not always means our model is perfectly accurate. We can perform more operations on the training dataset, tweak the parameters and check accuracy of the model again to make it production ready.

REFERENCES

- [1] <https://www.scipy.org/>
- [2] <https://numpy.org>
- [3] <https://pandas.pydata.org>
- [4] <https://matplotlib.org>
- [5] <https://seaborn.pydata.org>
- [6] <https://scikit-learn.org>



10.22214/IJRASET



45.98



IMPACT FACTOR:
7.129



IMPACT FACTOR:
7.429



INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Call : 08813907089  (24*7 Support on Whatsapp)