Census Income DataSet Classification

Sanket Kale

Abstract This is a classification problem to predict a person makes over 50k. It's a predictive modeling algorithm are evaluated based on their results. Classification is the process of predicting the class in the given dataset. Classes are nothing but targets or labels. To build the model we follow the following steps

Data preprocessing: transforming the raw data into an understandable format. The following are techniques we applied to our model to clean the data

- Dealing with missing values: replaced those values with the most frequent values.
- Dealing with Outliers: drop the data which are beyond that range
- Categorial Data: Convert the string data into numerical binary format 0 and 1.
- Label Encoder: encode labels with numeric distinct values, starting from 0 to n values.
- Scaling Data: It is a technique to standardize the feature data in a fixed range
- Handling imbalanced data with SMOTE (Oversampling technique)
- declare classifier with the flowing models and calculate the f1 Score
    - Logistic Regression
    - KNeighbors Classifier
    - Linear Discriminant Analysis
    - Gaussian NB
    - Ridge Classifier
    - Gradient Boosting Classifier
    - SVC
    - Random Forest Classifier

1 Introduction

Dataset : Census Income Dataset
(https://archive.ics.uci.edu/ml/datasets/Census+Income)
Extraction was done by Barry Becker from the 1994 Census database. A set of reasonably clean records was extracted using the following conditions: ((AAGE>16) && (AGI>100) && (AFNLWGT>1)&& (HRSWK>0))

Attributes Information

Columns	Description
age	continuous
workclass	Private, Self-emp-not-inc, Self-emp-inc, Federal-gov, Local-gov, State-gov, Without-pay, Never-worked.
fnlwgt	continuous.
education	Bachelors, Some-college, 11th, HS-grad, Prof-school, Assoc-acdm, Assoc-voc, 9th, 7th-8th, 12th, Masters, 1st-4th, 10th, Doctorate, 5th-6th, Preschool.
education-num	continuous.
marital-status	Married-civ-spouse, Divorced, Never-married, Separated, Widowed, Married-spouse-absent, Married-AF-spouse
occupation	Tech-support, Craft-repair, Other-service, Sales, Exec-managerial, Prof-specialty, Handlers-cleaners, Machine-op-inspct, Adm-clerical, Farming-fishing, Transport-moving, Priv-house-serv, Protective-serv, Armed-Forces.
relationship	Wife, Own-child, Husband, Not-in-family, Other-relative, Unmarried.
race	White, Asian-Pac-Islander, Amer-Indian-Eskimo, Other, Black.
sex	Female, Male.
capital-gain	continuous.
capital-loss	continuous.
hours-per-week	continuous.
native-country	United-States, Cambodia, England, Puerto-Rico, Canada, Germany, Outlying-US(Guam-USVI-etc), India, Japan, Greece, South, China, Cuba, Iran, Honduras, Philippines, Italy, Poland, Jamaica, Vietnam, Mexico, Portugal, Ireland, France, Dominican-Republic, Laos, Ecuador, Taiwan, Haiti, Columbia, Hungary, Guatemala, Nicaragua, Scotland, Thailand, Yugoslavia, El-Salvador, Trinadad&Tobago, Peru, Hong, Holand-Netherlands.
Income	>50K, <=50K. (Traget column)

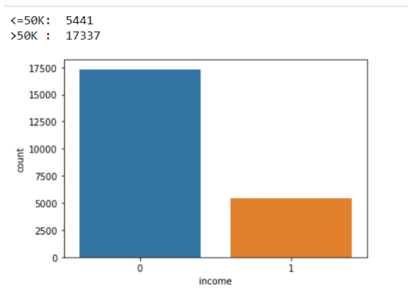
This is a classification problem and we should predict the below

Prediction task is to determine whether a person makes over 50K a year..

Motivation : This dataset has precise information about the people and their carrer background. It has the exact columns which we need to predict the people who can earn more than 50k per year.

2 Research

Classification problems are highly deal with imbalanced data. First, understand what is imbalance data means. For example, in the Census income dataset, we have two different classifications. One is <=50 and the second is >50. Total number of records 22,778 and if calculate and divide the record then <=50 contains 17,337 and >50 contains 5,441 records



```
<=50K:  5441
>50K :  17337
```

For this problem, we need to build a model with good performance with the minority class. There are different techniques to handle this problem such as oversampling and under-sampling. Oversampling means randomly duplicate the records in the minority class and under-sampling means randomly delete the records from the minority class.

In part one we applied the SMOTE technique to make the balance data. There different techniques of oversampling the data such as Random oversampling of the minority class, SMOTE, Borderline-SMOTE1, Borderline-SMOTE2, SMOTE + TOMEK, SMOTE+ENN

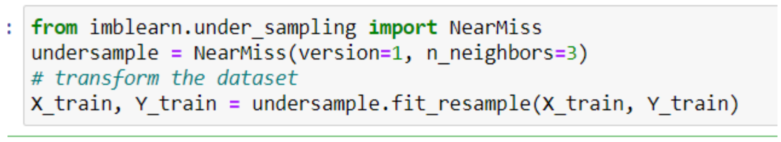
In this research, we are discussing the NearMiss technique.
NearMiss is the under-sampling technique is to make the majority class equal to the minority class.
There are three versions of NearMiss techniques, NearMiss1, NearMiss2, and NearMiss3
NearMiss-1: choose the minimum average distance to three closest minority class examples
NearMiss-2: choose the minimum average distance to three furthest minority class examples.
NearMiss-3: choose the minimum distance to each minority class example.

We can implement all 3 techniques and calculate performance

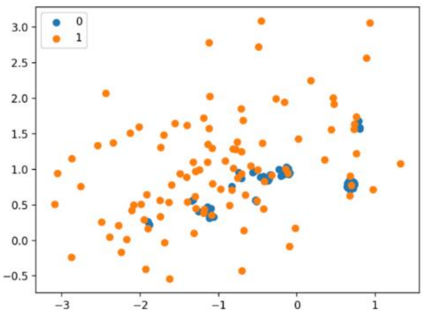
2.1 NearMiss-1

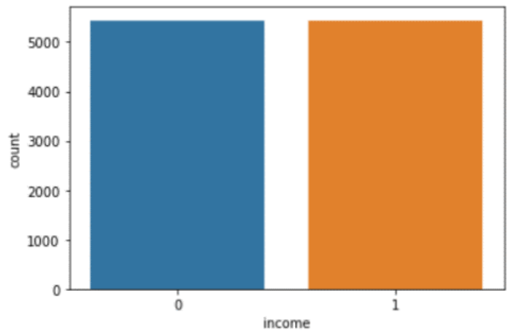
This is a by default technique will transform the majority class into the same minority class and selects those majority class that has minimum distance to 3 majority class
Parameter : Version = 1 ,n_neighbors=3

Before Sampling	5441	17337
After Sampling	5441	5441



```
from imblearn.under_sampling import NearMiss
undersample = NearMiss(version=1, n_neighbors=3)
# transform the dataset
X_train, Y_train = undersample.fit_resample(X_train, Y_train)
```

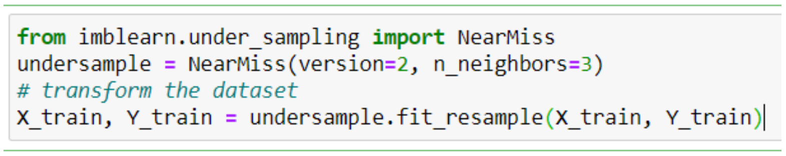




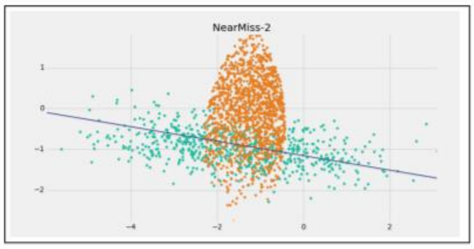
2.2 NearMiss-2

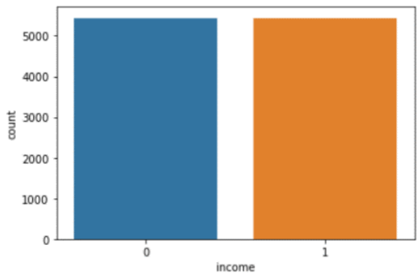
This exactly reverse of NearMiss1
This is a by default technique will transform the majority class into the same minority class and selects those majority class that has the closest distance to 3 majority class
Parameters : Version = 2 ,n_neighbors=3

Before Sampling	5441	17337
After Sampling	5441	5441



```
from imblearn.under_sampling import NearMiss
undersample = NearMiss(version=2, n_neighbors=3)
# transform the dataset
X_train, Y_train = undersample.fit_resample(X_train, Y_train)
```

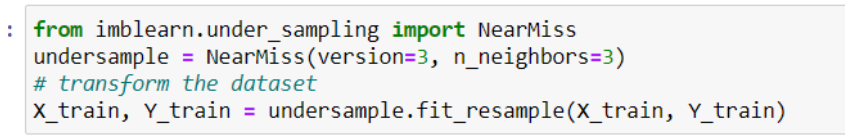




2.3 NearMiss-3

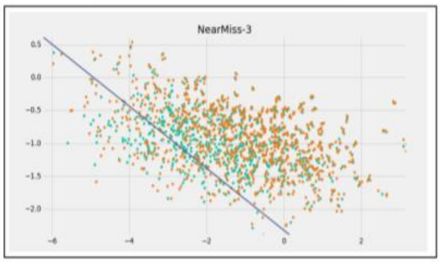
Selects the closest example from the majority of class

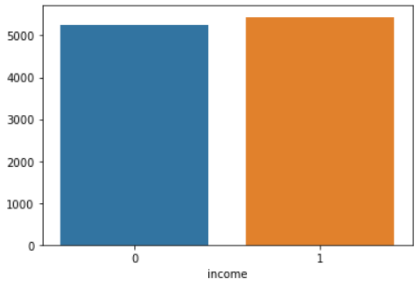
Parameters : Version = 2 ,n_neighbors=3



```
from imblearn.under_sampling import NearMiss
undersample = NearMiss(version=3, n_neighbors=3)
# transform the dataset
X_train, Y_train = undersample.fit_resample(X_train, Y_train)
```

Before Sampling	5441	17337
After Sampling	5259	5441





Methodology

3.1. Establishing a baseline

The following are the steps to build the model

- **Load the data** (Census income data)
- **Data Preprocessing**

  Handling with missing data: In the dataset description, it clearly mentioned that the missing values are in? , for cleaning data we need to replace those values with the most frequent values

  First, replace the ? with null values

  Example :
  data.replace("?", np.nan, inplace = True)
  data.isnull().sum()
  data_man["workclass"].replace(np.nan, workclass_mode, inplace = True)

- **Dealing with outlier**

  Some values are not related to other values. Only the fnlwgt column has an outlier. Drop the columns which are above the values 900000

  Example :
  data.drop(data[data["fnlwgt"]>900000].index,inplace=True)

- **Categorical Data**

  Convert the string data into the numeric format. The following are the columns sex, income, race, relationship, marital-status

  Marital-status and education columns are not required to build the model, so we dropped the columns from the data

  Example :
  data['sex'] = data['sex'].map({'Male': '1', 'Female': '0'})

- **LabelEncoder**

  We have 3 columns to apply the encoder such as workclass, occupation, native – country
  Fit and transform the data
  Example :
  for l in labels:

  data[l]=le.fit_transform(data[l])

- **Scaling Data :**

  This technique is used to standardize the range of features of data. Using StandardScaler fit and transform the data.

  Example :
  X = StandardScaler().fit_transform(data.loc[:, data.columns != 'income'])

  - **Split the data into the train and test data**

  X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.3, random_state = 0)

- **Handling imbalanced data**

  Using the oversampling and undersampling technique we can make equal the class of dataset.
  Here we used the oversampling technique such as SMOTE (**Synthetic Minority Oversampling Technique)**

  sm = SMOTE(random_state=0)
  X_train, Y_train = sm.fit_sample(X_train, Y_train)

  After cleaning the data we declare the classifier to build the model.

3.2 Hyperparameter Optimization

Hyperparameter: Hyperparameter, is the configuration that is external to the model and the value cannot be estimated from data. Specify the model parameter manually, that is model hyperparameter.

There are two strategies for hyperparameter tuning:

- GridSearchCV: It searches for the best hyperparameter from the grid.
- RandomizeSearchCV: It sets the hyperparameter in the grid in a random fashion and avoids unnecessary computation.

Here we have used the RandomizeSearchCV

Following hyperparameter used :

```
random_grid = {'n_estimators': n_estimators,
              'max_features': max_features,
              'max_depth': max_depth,
              'min_samples_split': min_samples_split
                         }
```

n_estimators: The number of trees
max_features: The number of features is considered while the best split
max_depth: The maximum depth of the tree
min_samples_split: The mini number of samples required to split an internal node

Hyperparameter is just a running of trails in a single job.

We applied hyperparameter on three models to increase the accuracy of the base model :

1. **Random forest Classifier**

**rf = RandomForestClassifier()
rf_random = RandomizedSearchCV(estimator = rf,scoring= "f1", param_distributions = random_grid, n_iter = 10, cv = 3, verbose=2, random_state=42, n_jobs = -1,)
rf_random.fit(X_train, Y_train)**

1. **GradientBoosting Classifier**

**rf = GradientBoostingClassifier()
rf_random = RandomizedSearchCV(estimator = rf,scoring= "f1", param_distributions = random_grid, n_iter = 10, cv = 3, verbose=2, random_state=42, n_jobs = -1,)
rf_random.fit(X_train, Y_train)**

1. **LogisticRegression Classifier**

lr = LogisticRegression(penalty = 'l2', C = i,random_state = 0)

lr.fit(X_train,Y_train)

3.2 Basic Implementation

The Basic Implementation is the same process which we use to build the model in part 1 but we used a different technique for Scaling and encoder

Encoder :
onehot encoder

The onehot encoder takes the categorical data and splits the column into multiple columns and the numbers are replaced by 0 and 1s

Using dummies – It allows encoding of columns using prefix. That will be easier for the remaining analysis with the proper naming conventions.

```
data=pd.get_dummies(data, columns=['workclass'], drop_first=True)
data=pd.get_dummies(data, columns=['occupation'], drop_first=True)
data=pd.get_dummies(data, columns=['native-country'], drop_first=True)
```

Scaling data: MinMaxScaler – rescale the variables in the range of 0,1

```
X = preprocessing.MinMaxScaler().fit_transform(data.loc[:, data.columns !=
'income'])
Y = data['income']
```

Feature selection: while building a model it helps to reduce the number of input variables. It is the process of ranking each feature of the dataset to predict the target regression value of the class.

SelectPercentile : It gives statistical test between feature data and target data

```
a = SelectPercentile(chi2, percentile=10).fit(X_train, Y_train)
```

4 Evaluation

We have calculate the f1 score to evaluate the performance of the model

f1 score : combines the precision and recall relative to specific positive class

f1 = 2 * (precision * recall) / (precision + recall)

The below are the results which we got into all three parts'

Model	Baseline f1 score	f1 score with feature selection	f1 score with hyper parameter
LogisticRegression	0.6743	0.62	0.6744
KNeighborsClassifier	0.6535	0.61	
LinearDiscriminantAnalysis	0.6418	0.61	
GaussianNB	0.5793	0.52	
RidgeClassifier	0.6418	0.61	
GradientBoostingClassifier	0.7031	0.62	0.7186
SVC	0.6694	0.62	
RandomForestClassifier	0.6923	0.63	0.6946

As we got the f1 score for all the models. If you compare the results of the hyperparameter tunning gives a bit high accuracy than basic implementation.

Now if we implement the feature selection and then compare the results we got less accuracy than the basic evaluation.

In the research section, we discussed the resampling technique to handle imbalanced data. we performed three different techniques to evaluate the results and calculate the f1score, but we got less accuracy than the basic evaluation.

Logistic Regression			
	NearMiss1	NearMiss2	NearMiss3
f1 Score	0.6189	0.6285	0.6702

In the basic evaluation, we used the SMOTE which is the oversampling technique to handle imbalanced data. we got a better result by using SMOTE, where we transform the minority samples into majority samples and balance the data. For our dataset, we have to use SMOTE rather than using the NearMiss technique.

5 Conclusion

To conclude this article, we applied different techniques for **preprocessing** the data and build a model in Part 1 A that gives a good performance. We applied **hyperparameter tuning** on three models to increase the f1 score. Hyperparameter tuning is nothing but searching for the ideal model architecture. Then we used the **onehot encoding** and **SelectPercentile** a feature selection technique to improve f1 score but for our dataset it this technique was not suitable. In the research area, we proposed the **NearMiss** technique (undersampling method) to handle the imbalanced data. This technique decreases the examples of majority class into minority class. There are three versions of the NearMiss (NearMiss1, NearMiss2, NearMiss3). We applied all three techniques but that gave comparatively less score than the baseline model.
In the baseline, we used SMOTE (oversampling method) which gave us a better score than NearMiss.

Further, we can apply the different feature selection techniques such as the SelectKBest technique to improve the accuracy of the model.

References [1]

1. https://machinelearningmastery.com/undersampling-algorithms-for-imbalanced-classification/
2. https://arxiv.org/pdf/1608.06048.pdf