Sampling

Dr. Ernesto Lee

Where would you Encounter Imbalanced Data

Some of the use cases where we encounter imbalanced datasets include the following:

- Fraud detection for credit cards or insurance claims
- Medical diagnoses where we must detect the presence of rare diseases
- Intrusion detection in networks

NOTE: The bank dataset has 88% NO and 12% YES

Business Context

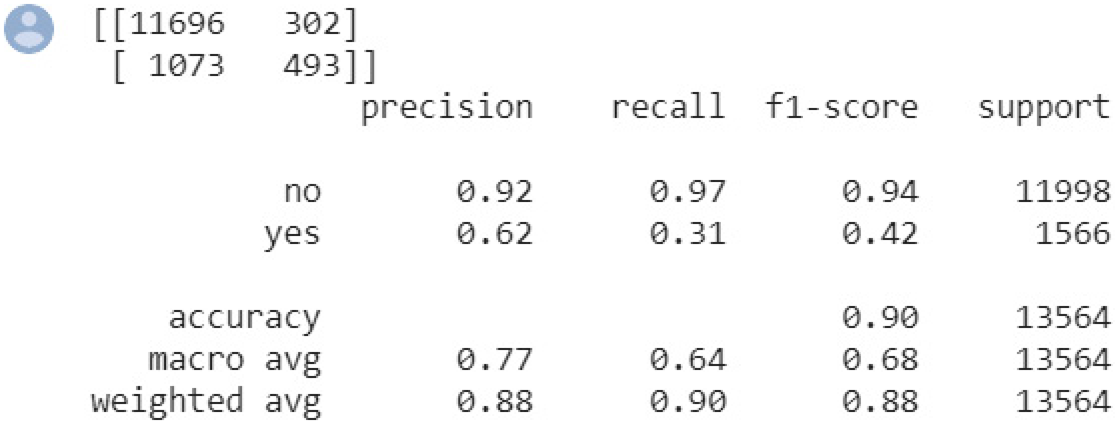
- It has been observed that a large proportion of customers who were identified as potential cases for targeted marketing for term deposits have turned down the offer.
- This has made a big dent in the sales team's metrics on upselling and cross-selling.
- The business team urgently requires your help in fixing the issue to meet the required sales targets for the quarter.

Benchmark the Model

- Perform an EDA on the data
- Generate a Logistic Regression Model
- Analyze the baseline

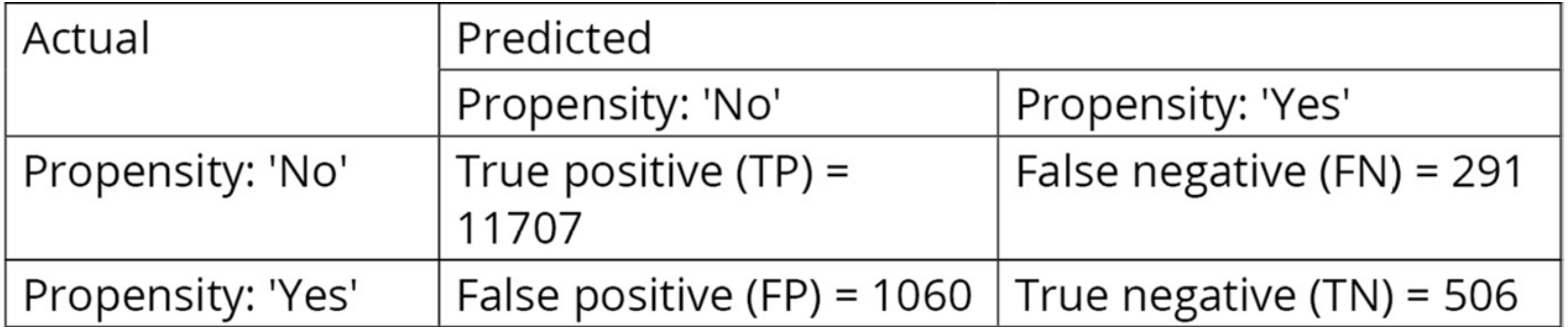
DEMO

Baseline Analysis



```
[[11696   302]
 [ 1073   493]]
              precision    recall  f1-score   support

          no       0.92      0.97      0.94     11998
         yes       0.62      0.31      0.42      1566

    accuracy                           0.90     13564
   macro avg       0.77      0.64      0.68     13564
weighted avg       0.88      0.90      0.88     13564
```

Analysis



Actual	Predicted	
	Propensity: 'No'	Propensity: 'Yes'
Propensity: 'No'	True positive (TP) = 11707	False negative (FN) = 291
Propensity: 'Yes'	False positive (FP) = 1060	True negative (TN) = 506

Accuracy

In our case, it will be (11707 + 506) / (11707 + 1060 + 291 + 506), or 90%.

$$Accuracy\ of\ a\ model = \frac{(\mathrm{TP} + \mathrm{TN})}{(\mathrm{TP} + \mathrm{FP} + \mathrm{FN} + \mathrm{TN})}$$

Precision

In our case, for the positive class, the precision is *TP/(TP + FP)*, which is 11707/ (11707 + 1060), which comes to approximately 92%.

In the case of the negative class, the precision could be written as *TN / (TN + FN)*, which is 506 / (506 + 291), which comes to approximately 63%.

$$Precision\ value = \frac{Correct\ prediction\ of\ the\ class}{Total\ predictions\ for\ that\ class}$$

Recall

The recall value for the positive class, *TP / (TP + FN)* = 11707 / (11707 + 291), comes to approximately 98%.

The recall value for the negative class, *TN / (TN + FP)* = 506 / (506 + 1060), comes to approximately 32%.

$$Recall\ value = \frac{Correct\ prediction\ of\ the\ class}{Total\ examples\ of\ the\ class}$$

Bias…

Why do you think that the classifier is biased toward one class?

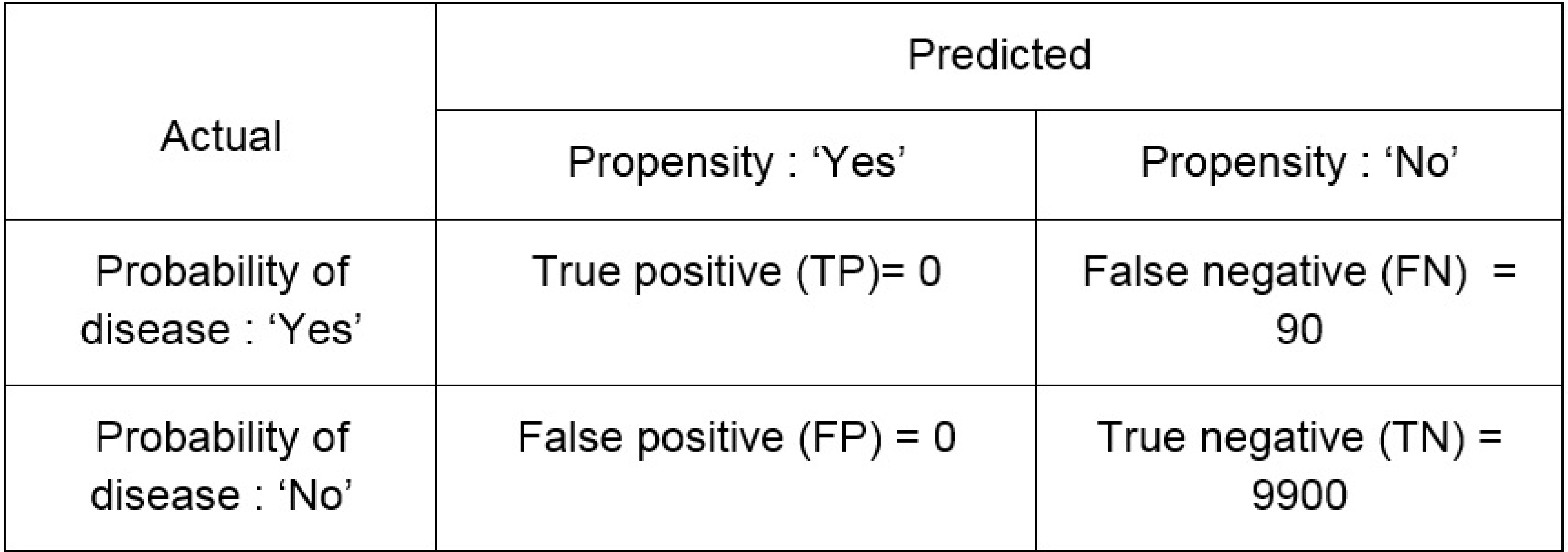
The answer to this can be unearthed by looking at the class balance in the training set.

DEMO

Why Accuracy is a Bad Metric with Imbalanced Data

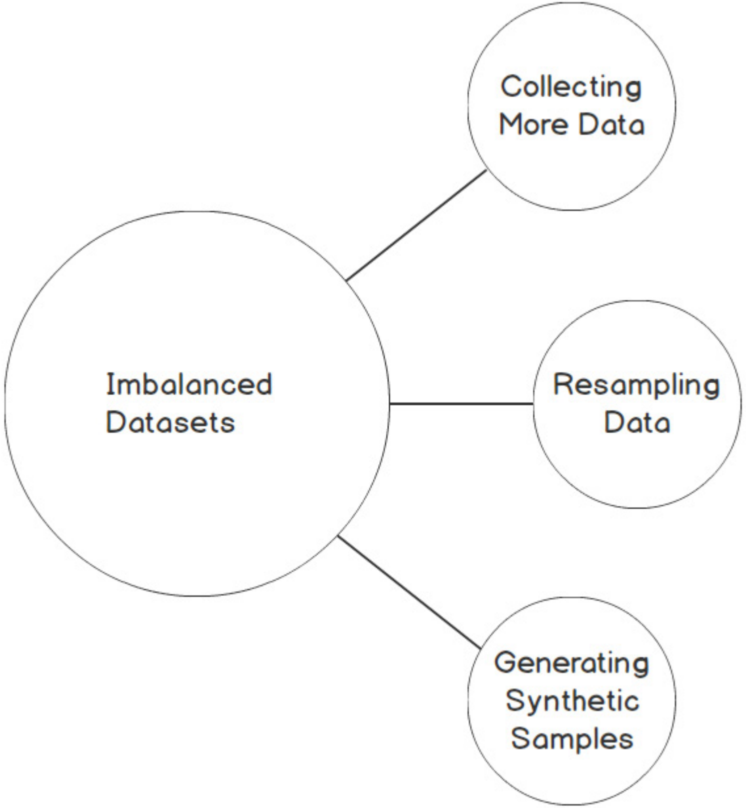
a dataset where the negative class is around 99% and the positive class is 1% (as in a use case where a rare disease has to be detected, for instance).

Data set Size: 10,000 examples



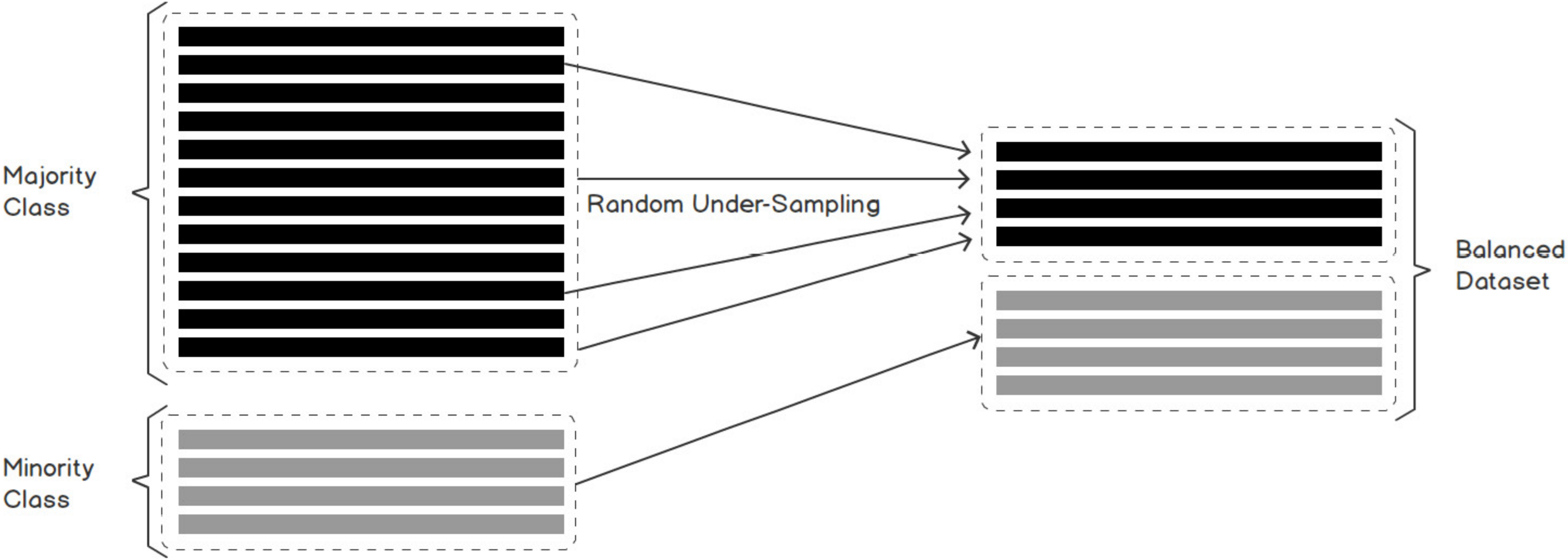
Actual	Predicted	
	Propensity : ‘Yes’	Propensity : ‘No’
Probability of disease : ‘Yes’	True positive (TP)= 0	False negative (FN) = 90
Probability of disease : ‘No’	False positive (FP) = 0	True negative (TN) = 9900

How Do you Deal with Imbalanced Data?

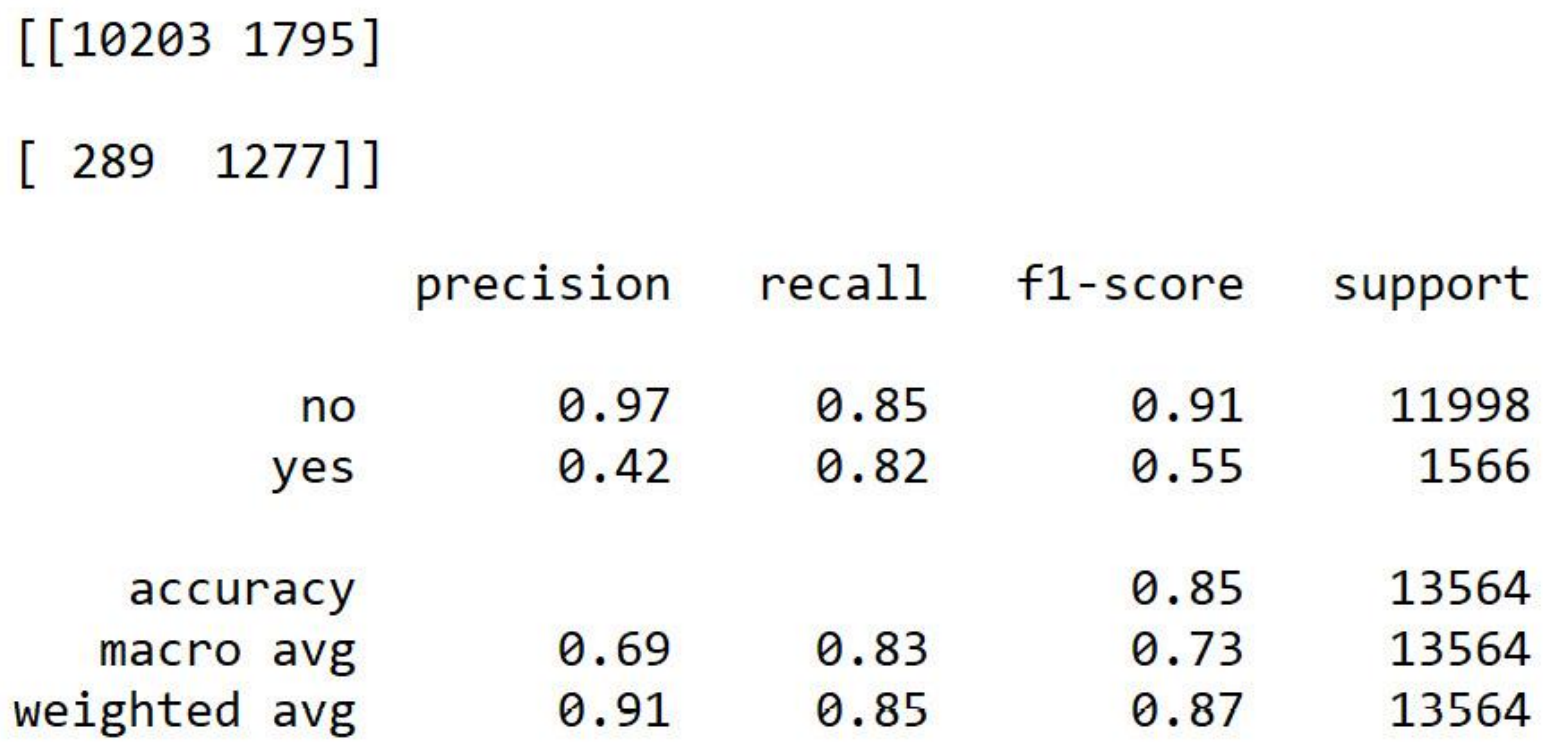


UnderSampling

Demo

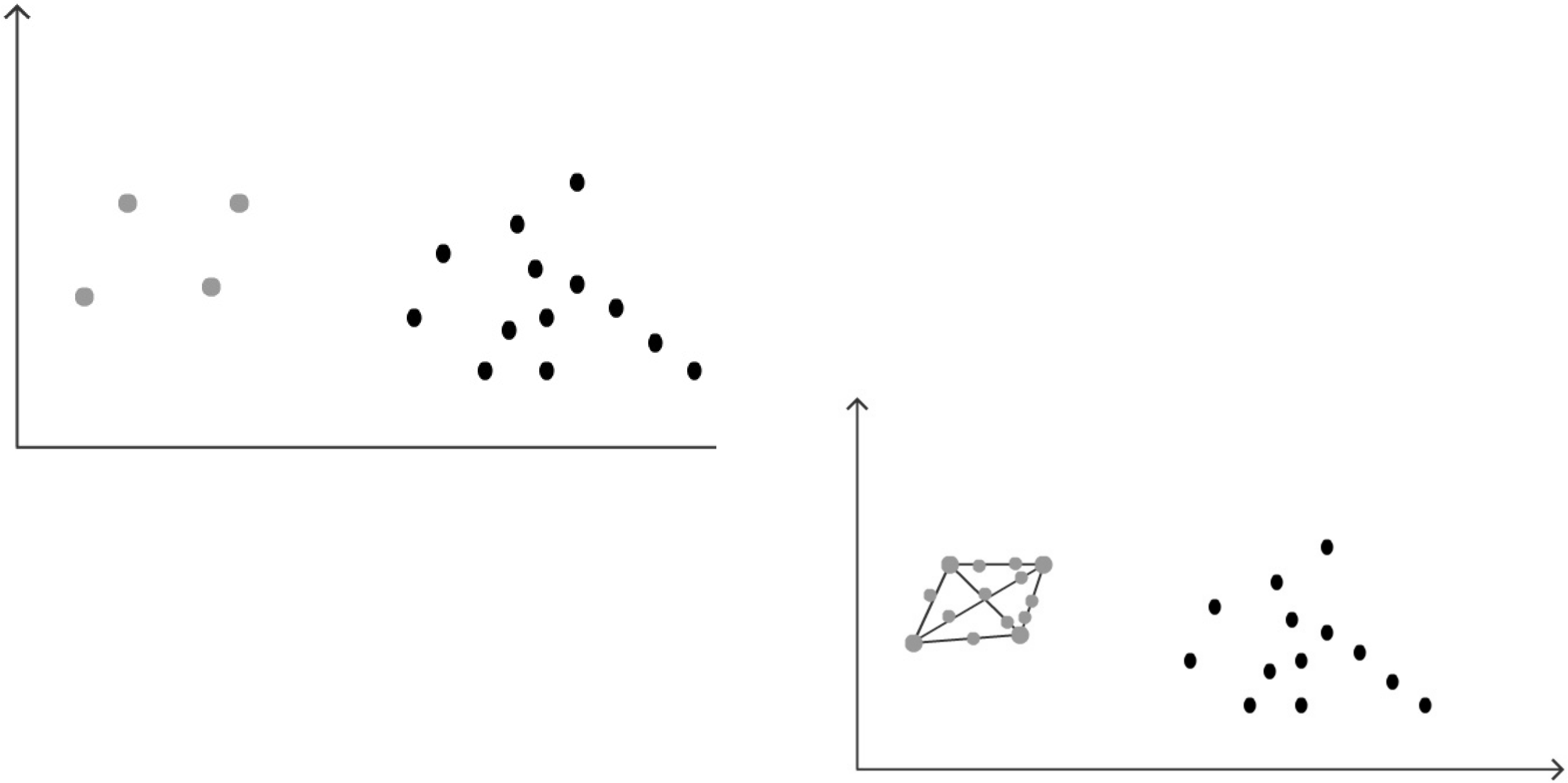


UnderSampling Analysis

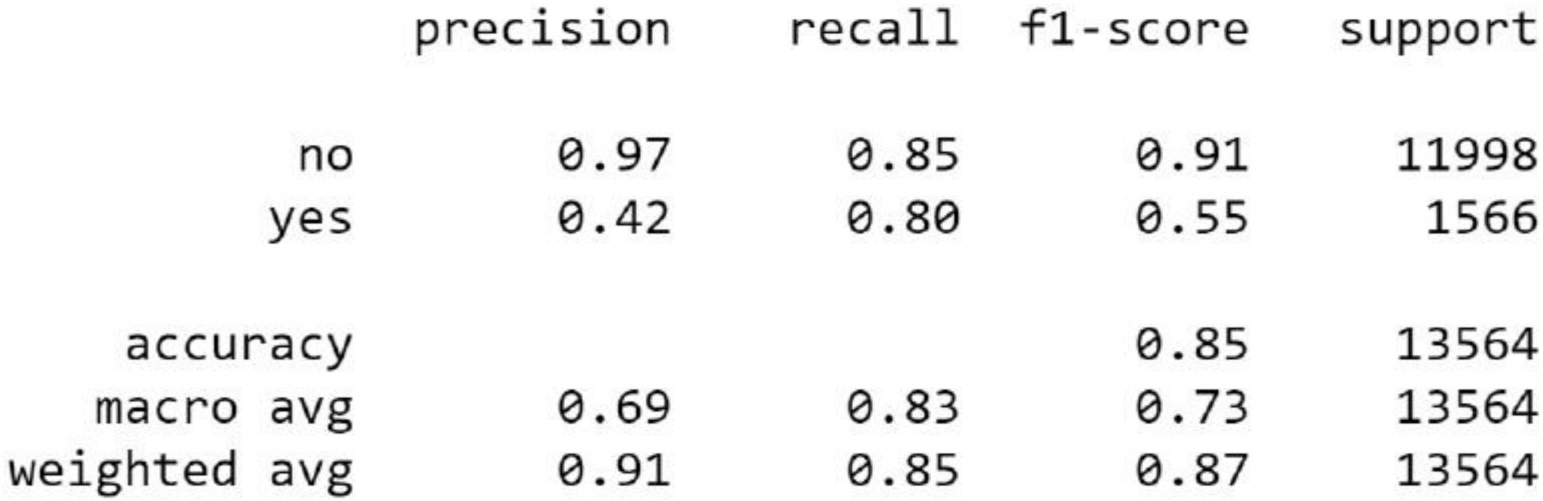


```
[[10203 1795]

 [ 289  1277]]

              precision    recall  f1-score   support

          no       0.97      0.85      0.91     11998
         yes       0.42      0.82      0.55      1566

    accuracy                           0.85     13564
   macro avg       0.69      0.83      0.73     13564
weighted avg       0.91      0.85      0.87     13564
```

SMOTE: Oversampling (demo)



SMOTE Analysis



```
              precision    recall  f1-score   support

          no       0.97      0.85      0.91     11998
         yes       0.42      0.80      0.55      1566

    accuracy                           0.85     13564
   macro avg       0.69      0.83      0.73     13564
weighted avg       0.91      0.85      0.87     13564
```

Group Lab

1. Implement all the initial steps, which include installing smote-variants and loading the data using pandas. churn.csv
2. Normalize the numerical raw data using the MinMaxScaler() function we learned about in class, Binary Classification.
3. Create dummy data for the categorical variables using the pd.get_dummies() function.
4. Separate the numerical data from the original data frame.
5. Concatenate numerical data and dummy categorical data using the pd.concat() function.
6. Split the earlier dataset into train and test sets using the train_test_split() function.
7. Since the dataset is imbalanced, you need to perform the various techniques mentioned in the following steps.
8. For the undersampling method, find the index of the minority class using the .index() function and separate the minority class. After that, sample the majority class and make the majority dataset equal to the minority class using the .sample() function. Concatenate both the minority and under-sampled majority class to form a new dataset. Shuffle the dataset and separate the X and Y variables.
9. Fit a logistic regression model on the under-sampled dataset and name it churnModel1.
10. For the SMOTE method, create the oversampler using the sv.SMOTE() function and create the new X and Y training sets.
11. Fit a logistic regression model using SMOTE and name it churnModel2.
12. Generate the two separate predictions for each model.
13. Generate separate accuracy metrics, classification reports, and confusion matrices for each of the predictions.
14. Analyze the results and select the best method.