SUPERVISED MACHINE LEARNING

CLASSIFICATION

TING CHONG NA
28.05.2023

Data Description

TOPIC: Predicting Customer Churn

Objective: Predicting customer churns (leaving the business) of a telecom company

This dataset is processed and contains **42 features** about a customer's telcom service types, tenure, charges, and payments. Based on such features, we would like to predict if a customer is leaving the business or not (churn).

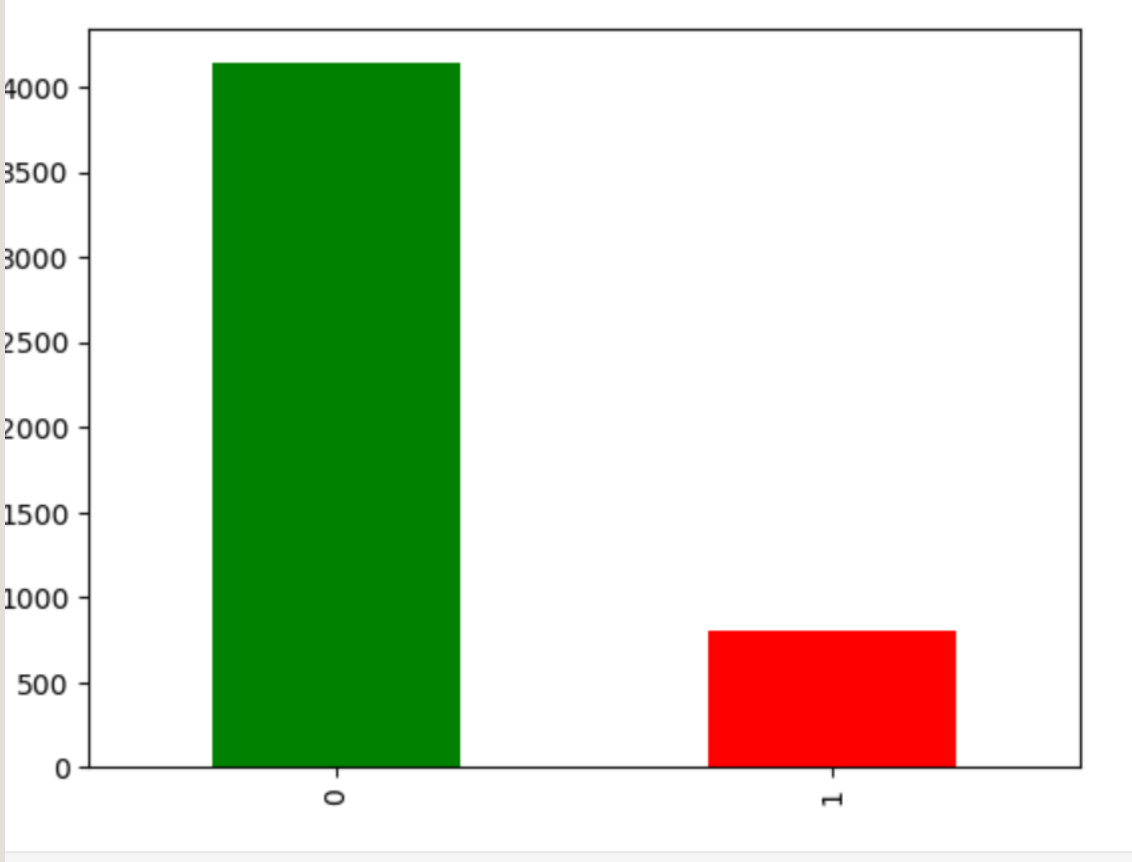
```
churn_df.head()
```

	tenure	MonthlyCharges	TotalCharges	Partner_0	Partner_1	Dependents_0	Dependents_1	PhoneService_0	PhoneService_1	Multiple
0	27	70.55	1943.90	1.0	0.0	1.0	0.0	0.0	1.0	
1	69	93.30	6398.05	1.0	0.0	0.0	1.0	0.0	1.0	
2	55	59.20	3175.85	0.0	1.0	1.0	0.0	0.0	1.0	
3	49	59.60	2970.30	1.0	0.0	0.0	1.0	0.0	1.0	
4	72	109.55	7887.25	1.0	0.0	0.0	1.0	0.0	1.0	

5 rows × 43 columns

y=0 : Non-Churn
y=1 : Churn

Data Analysis




```
y_train.value_counts()
```

```
0    4139
1     800
Name: Class, dtype: int64
```

An Imbalanced Dataset

The non-churn customers (4139) are almost 4 times more than the churn customers (800)

Model: Random Forest Classifier

Train a regular random forest classifier without any add-ons (class weights or resampling)

```
def split_data(df):
    X = df.loc[ : , df.columns != 'Class']
    y = df['Class'].astype('int')
    return train_test_split(X, y, test_size=0.2, stratify=y, random_state = rs)
```

Train Test Split

```
def build_rf(X_train, y_train, X_test, threshold=0.5, best_params=None):

    model = RandomForestClassifier(random_state = rs)
    # If best parameters are provided
    if best_params:
        model = RandomForestClassifier(random_state = rs,
                                       # If bootstrap sampling is used
                                       bootstrap = best_params['bootstrap'],
                                       # Max depth of each tree
                                       max_depth = best_params['max_depth'],
                                       # Class weight parameters
                                       class_weight=best_params['class_weight'],
                                       # Number of trees
                                       n_estimators=best_params['n_estimators'],
                                       # Minimal samples to split
                                       min_samples_split=best_params['min_samples_split'])
    # Train the model
    model.fit(X_train, y_train)
    # If predicted probability is largr than threshold (default value is 0.5), generate a positive label
    predicted_proba = model.predict_proba(X_test)
    yp = (predicted_proba [:,1] >= threshold).astype('int')
    return yp, model
```

Random Forest Classifier Model

Robust Evaluation Metric For binary classification

Precision
- Percentage of accurately predicted positive instance

Recall
- Percentage of successfully recognized positive instances

Fscore
- Weighted Average of precision and recall to evaluate the model

	Actual Positive	Actual Negative
Predicted Positive	True Positive (TP)	False Positive (FP)
Predicted Negative	False Negative (FN)	True Negative (FN)

- Precision = TP / (TP + FP)
- Recall = TP / (TP + FN)
- F-score: weighted average of Precision and Recall

Model: Random Forest Classifier

```
preds, model = build_rf(X_train, y_train, X_test, best_params=best_params_no_weight)
result = evaluate(y_test, preds, "Original")
print(result)
results.append(result)
```

```
{'type': 'Original', 'accuracy': 0.8623481781376519, 'recall': 0.28, 'auc': 0.6274396135265701, 'precision': 0.682926829268292
7, 'fscore': 0.2865013774104683}
```

RESULTS:
High Accuracy (0.86)
Low Recall (0.28)

To Improve Performance:
1. Add class re-weighting
2. **Resampling: SMOTE and Undersampling**

Model: **Add class re-weighting**

Add class weights to the RFC with pre-tuned weight 0.8 to churn class and weight 0.2 to non-churn class

```
class_weight = {}
# 0.2 to Non-churn class
class_weight[0] = 0.2
# 0.8 to Churn class
class_weight[1] = 0.8
```

```
# Pre-tuned hyper parameters
best_params_weight = {'bootstrap': True,
                      'class_weight': class_weight,
                      'max_depth': 10,
                      'min_samples_split': 5,
                      'n_estimators': 50}
```

Evaluate the refined model

```
# class weight
preds_cw, weight_model = build_rf(X_train, y_train, X_test, best_params=best_params_weight)
```

```
result = evaluate(y_test, preds_cw, "Class Weight")
print(result)
results.append(result)
```

```
{'type': 'Class Weight', 'accuracy': 0.8137651821862348, 'recall': 0.62, 'auc': 0.7356038647342995, 'precision': 0.44604316546
76259, 'fscore': 0.6108374384236454}
```

Model: **Add class re-weighting**

```
results
```

```
[{'type': 'Original',
  'accuracy': 0.8623481781376519,
  'recall': 0.28,
  'auc': 0.6274396135265701,
  'precision': 0.6829268292682927,
  'fscore': 0.2865013774104683},
 {'type': 'Class Weight',
  'accuracy': 0.8137651821862348,
  'recall': 0.62,
  'auc': 0.7356038647342995,
  'precision': 0.4460431654676259,
  'fscore': 0.6108374384236454}]
```

IMPROVEMENT
Recall increased from 0.28 to 0.62
Fscore increased from 0.29 to 0.61

Add Class Reweighting is effective for the imbalanced customer churn dataset.

Model: Resampling SMOTE and Undersampling

Use resampling SMOTE and undersampling

```
# X_smo is resampled from X_train using SMOTE
# y_smo is resampled from y_train using SMOTE
# X_under is resampled from X_train using Undersampling
# y_under is resampled from y_train using Undersampling
X_smo, y_smo, X_under, y_under = resample(X_train, y_train)
```

```
def resample(X_train, y_train):
    # SMOTE sampler (Oversampling)
    smote_sampler = SMOTE(random_state = 123)
    # Undersampling
    under_sampler = RandomUnderSampler(random_state=123)
    # Resampled datasets
    X_smo, y_smo = smote_sampler.fit_resample(X_train, y_train)
    X_under, y_under = under_sampler.fit_resample(X_train, y_train)
    return X_smo, y_smo, X_under, y_under
```

Evaluate the refined model

```
preds_smo, smo_model = build_rf(X_smo, y_smo, X_test, best_params=best_params_no_weight)
result = evaluate(y_test, preds_smo, "SMOTE")
print(result)
results.append(result)
```

```
{'type': 'SMOTE', 'accuracy': 0.8356275303643724, 'recall': 0.505, 'auc': 0.7022584541062802, 'precision': 0.4926829268292683,
'fscore': 0.5045148895292987}
```

```
preds_under, under_model = build_rf(X_under, y_under, X_test, best_params=best_params_no_weight)
result = evaluate(y_test, preds_under, "Undersampling")
print(result)
results.append(result)
```

```
{'type': 'Undersampling', 'accuracy': 0.7336032388663968, 'recall': 0.79, 'auc': 0.7563526570048309, 'precision': 0.3550561797
752809, 'fscore': 0.7544536271809001}
```

Model: Resampling SMOTE and Undersampling

```
results
```

```
[{'type': 'Original',
  'accuracy': 0.8623481781376519,
  'recall': 0.28,
  'auc': 0.6274396135265701,
  'precision': 0.6829268292682927,
  'fscore': 0.2865013774104683},
 {'type': 'Class Weight',
  'accuracy': 0.8137651821862348,
  'recall': 0.62,
  'auc': 0.7356038647342995,
  'precision': 0.4460431654676259,
  'fscore': 0.6108374384236454},
 {'type': 'SMOTE',
  'accuracy': 0.8356275303643724,
  'recall': 0.505,
  'auc': 0.7022584541062802,
  'precision': 0.4926829268292683,
  'fscore': 0.5045148895292987},
 {'type': 'Undersampling',
  'accuracy': 0.7336032388663968,
  'recall': 0.79,
  'auc': 0.7563526570048309,
  'precision': 0.3550561797752809,
  'fscore': 0.7544536271809001}]
```

IMPROVEMENT

<u>For Resampling SMOTE</u>

Recall increased to 0.505

Fscore increased to 0.50

<u>For Undersampling</u>

Recall increased to 0.79

Fscore increased to 0.75

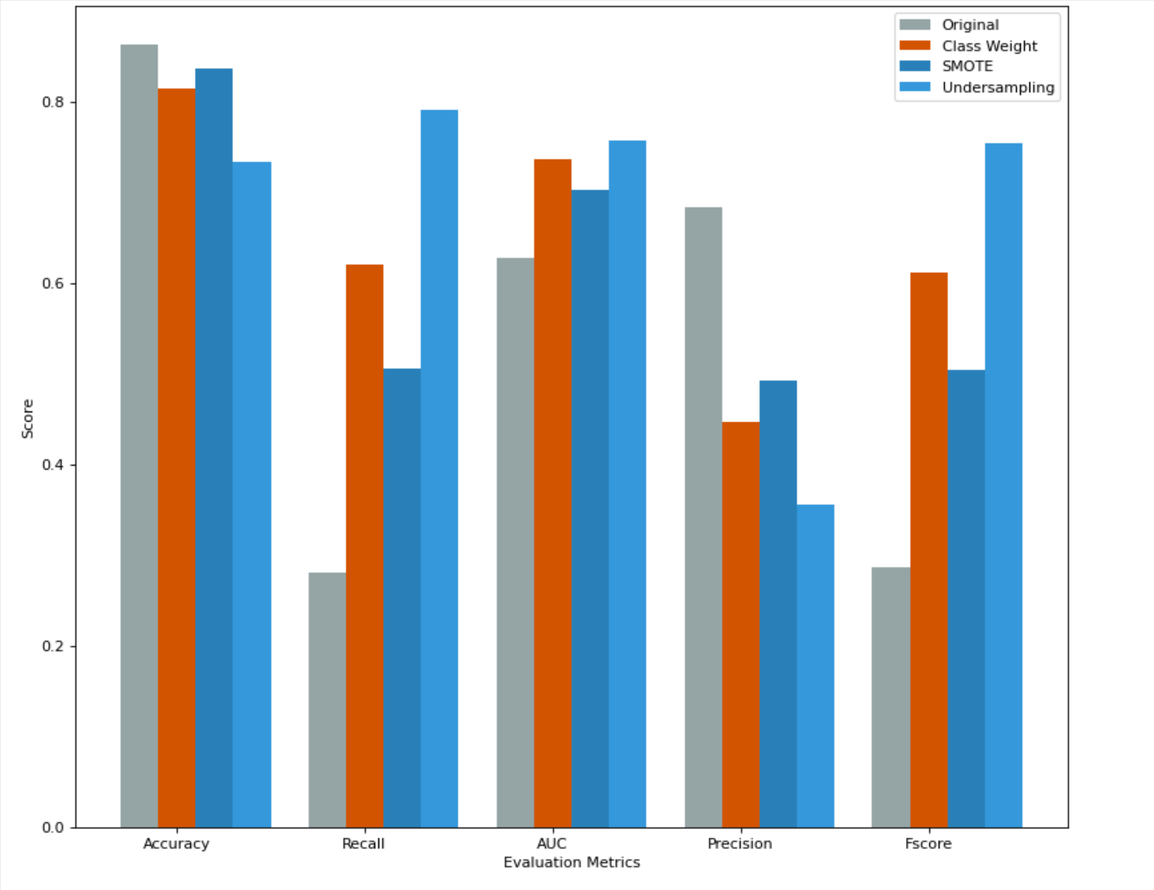
Resampling SMOTE and Undersampling is effective for the imbalanced customer churn dataset.

Model: Compare Results

Compare the performance among different random forest models:

- Model trained with Random Forest Classifier
- Model with Class Weights
- Model trained with Resampling SMOTE
- Model trained with Undersampling

Model: Compare Results



Accuracy
All models have high accuracy.

Recall
Improved with Class Weights, Resampling and Undersampling.
Undersampling produces the highest recall.

AUC
Improved with Class Weights, Resampling and Undersampling.
Undersampling produces the highest AUC.

Precisions
Decreased with Class Weights, Resampling and Undersampling (increased false positives)

Fscore
Improved with Class Weights, Resampling and Undersampling.
Undersampling has the highest Fscore.

Conclusion

By analyzing the bar chart above, **Undersampling** seems to be the **best method** to help alleviate the imbalanced challenge in the customer churn dataset.

Although all Class Weights, Resampling and Undersampling decreased the Precision (increased false positives) but sometimes it is not a bad idea to assume some customers are about to leave as motivation to improve services.