E-COMMERCE CUSTOMER CHURN

6 JANUARY, 2024

Zeynep Sena Tınaz
Erva Yurtbaş
Gizem Yüzer

LITERATURE REVIEW

E-commerce Customer Churn Prediction Based on Improved SMOTE and AdaBoost

Xiaojun Wu
School of Economics and Management
Tongji University
Shanghai,China
wxjun9999@tongji.edu.cn

Sufang Meng
School of Economics and Management
Tongji University
Shanghai,China
mengsf000@163.com

***Abstract*— E-commerce customer churn rate is high and the customer churn dataset is seriously imbalanced. In order to improve the prediction accuracy of churn customers as well as strengthen to identify non-churn customers, this paper presents e-commerce customer churn prediction model based on improved SMOTE and AdaBoost. First, processing the churn data with improved SMOTE, which combines oversampling and undersampling methods to address the imbalance problem and subsequently integrates AdaBoost algorithm to predict. Finally, the empirical study on B2C E-commerce platform proves that this model has better efficiency and accuracy compared with the mature customer churn prediction algorithms.**

Chun hua et al.[2] established an e-commerce customer churn prediction model combined with individual activity called H-ULSSVM. X Yu et al.[3] proposed an extended support vector machine (ESVM) by introducing parameters to tell the impact of churner, non-churner and nonlinear. Xie Y[4] proposed the improved balanced random forests (IBRF) and applied this method into a real bank customer churn data set. The nature of IBRF is that the best features are iteratively learned by altering the class distribution and by putting higher penalties on misclassification of the minority class. Tsai C F[5]considered two hybrid models by combining two different neural network techniques for churn prediction, which were back-propagation artificial neural networks (ANN) and self-organizing maps (SOM).

LITERATURE REVIEW

Send Orders for Reprints to reprints@benthamscience.ae

800 *The Open Cybernetics & Systemics Journal*, 2014, *8*, 800-804

Open Access

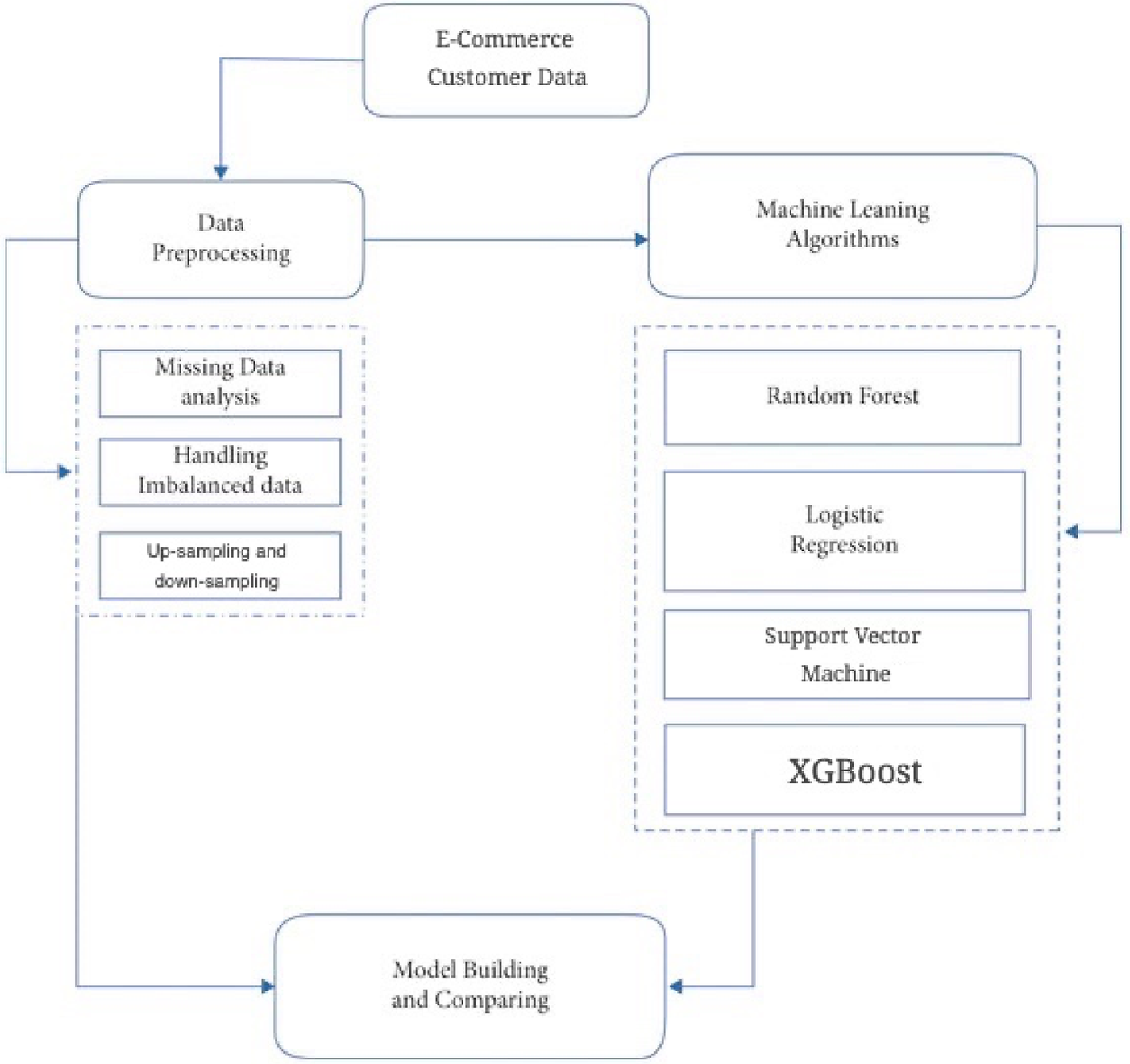
Research on E-Commerce Customer Churning Modeling and Prediction

Xue Zhao*

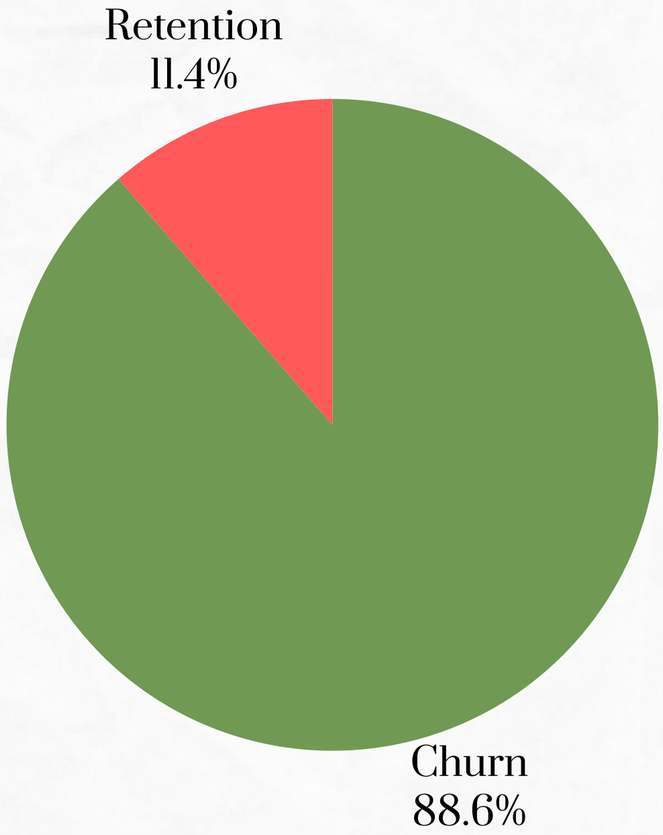
International Education College, Nanyang Institute of Technology, Nanyang, 473004, China

Abstract: This paper discusses the customer churning prediction problem in electronic commerce. In electronic commerce the customer data change is non-linear and time-varying and other characteristics, using a single prediction model to accurately predict e-commerce customer loss is difficult. In order to improve the prediction accuracy rate of electronic commerce churning, the model first uses the genetic algorithm for the screening of effecting factors, and extracts the important influence factors which affect the predicting results. Then support vector machine and neural network are respectively used to carry out the forecast. Finally, using support vector machine fuses the two prediction results to acquire the prediction results of the combination model. Simulation results show that the combined model can improve the prediction accuracy rate of the electronic commerce customer churning, and provides a new prediction method for the electronic commerce customer churning.

Keywords: Customer churning prediction, e-commerce, neural network (NN), support vector machine (SVM).


E-Commerce
Customer Data
Data
Preprocessing
Machine Leaning
Algorithms
Missing Data
analysis
Handling
Imbalanced data
Up-sampling and
down-sampling
Random Forest
Logistic
Regression
Support Vector
Machine
XGBoost
Model Building
and Comparing

ABOUT OUR DATASET



We have:

- 48 columns
- 49358 data entries:5647 of them are churn, 43711 of them are retention
- 1 target_class: 1(retention) 0(churn)
- So it’s a binary clasification problem

DATA PREPROCESSING

IMBALANCED DATA

We have an imbalanced data so we used some upsampling and downsampling methods:

UPSAMPLING

Churn Upsample

```
: from sklearn.utils import resample
churn_upsample = resample(churn,
             replace=True,
             n_samples=20000,
             random_state=42)

print(churn_upsample.shape)
```

```
(20000, 48)
```

By duplicating existing churn data we have increased number of churn data to 20000

Downsampling

By removingsome retention data we create an balanced data

Retention Downsample

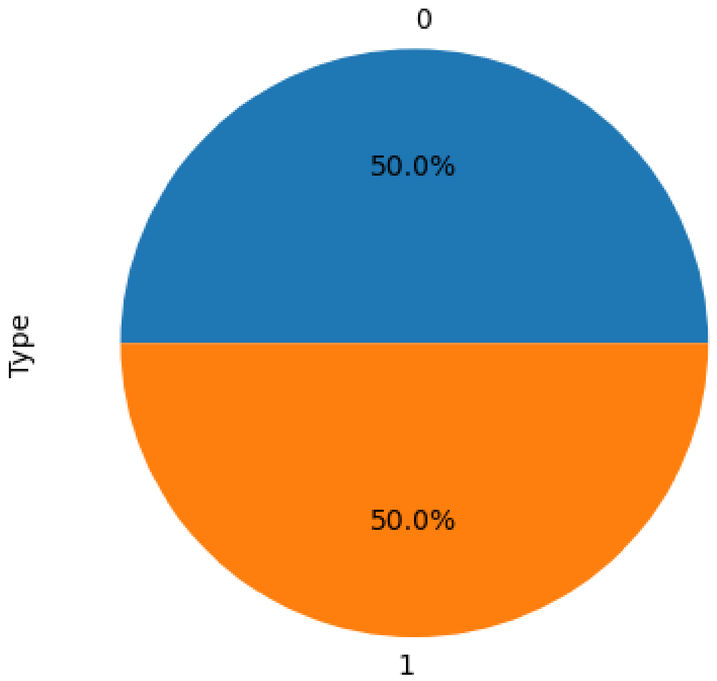
```
from sklearn.utils import resample
retention_downsample = resample(retention,
             replace=True,
             n_samples=len(churn_upsample),
             random_state=42)

print(retention_downsample.shape)
```

```
(20000, 48)
```

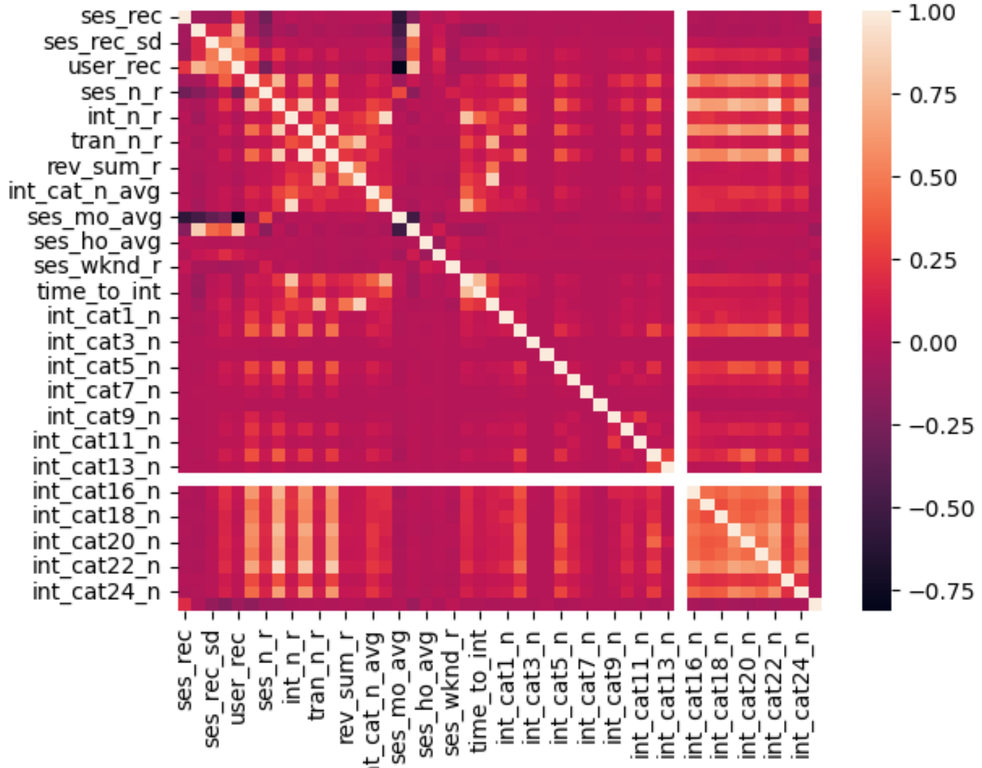
DATA PREPROCESSING

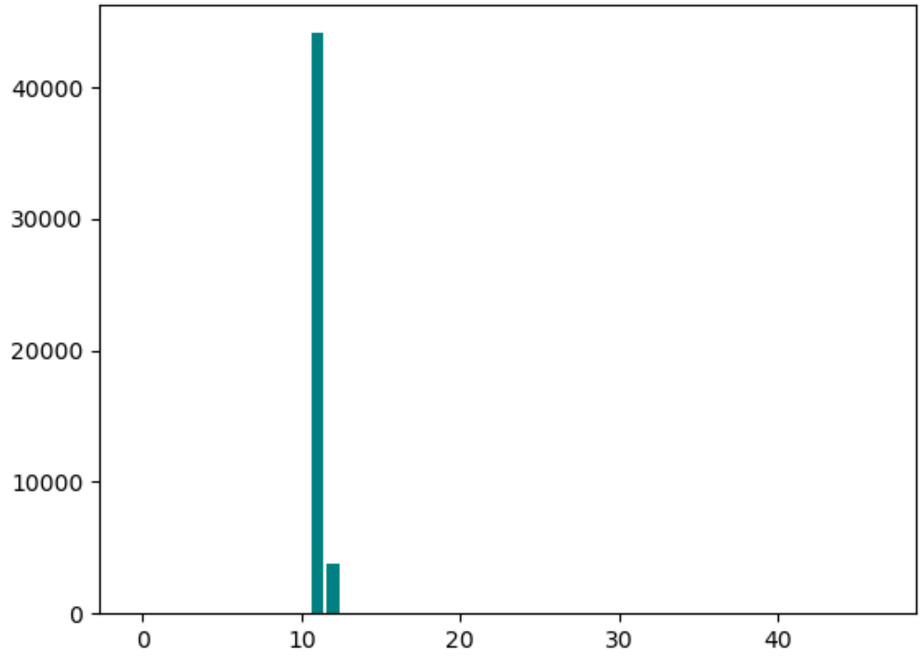
Upsampling and Downsampling (Together)

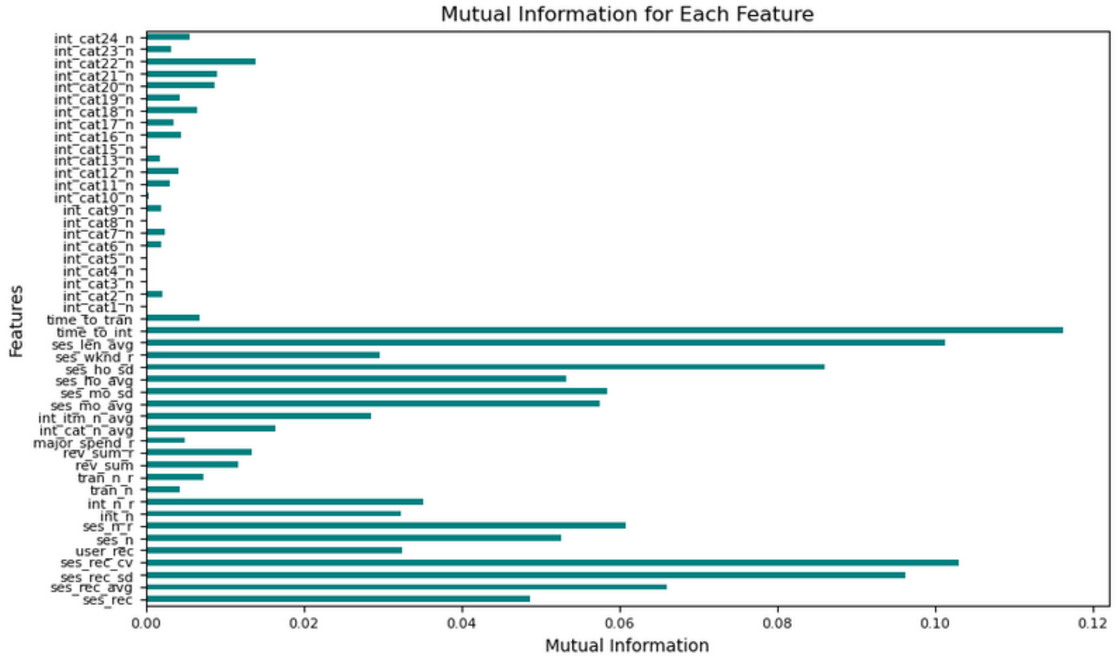
```
data_downsampled = pd.concat([retention_downsample, churn_upsample])

print(data_downsampled["target_class"].value_counts())

data_downsampled.groupby('target_class').size().plot(kind='pie',
                                        y = "target_class",
                                        label = "Type",
                                        autopct='%1.1f%%')
```



EXPLOTARY DATA ANALYSIS



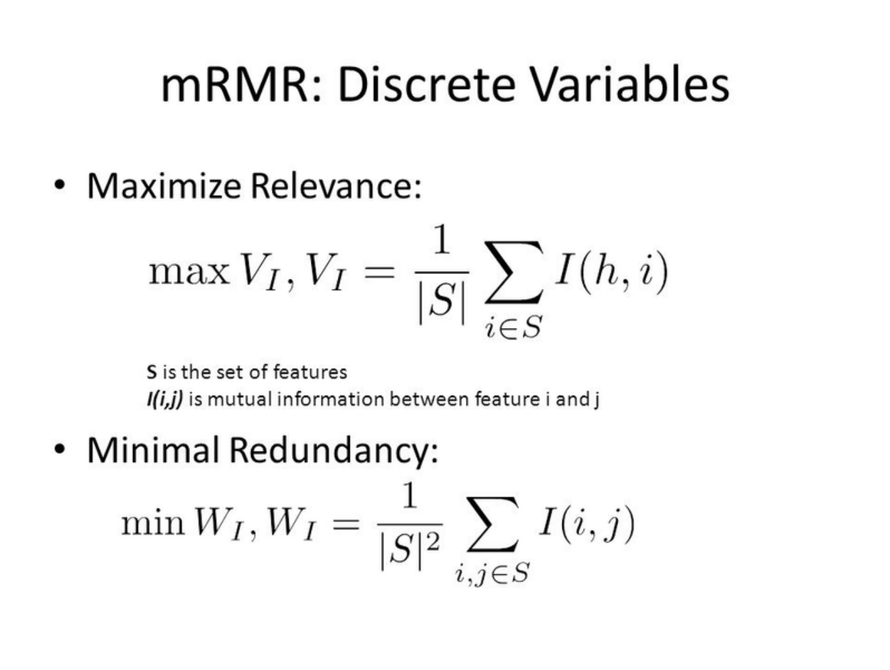




FEATURE SELECTION

Feature Selection with MRMR

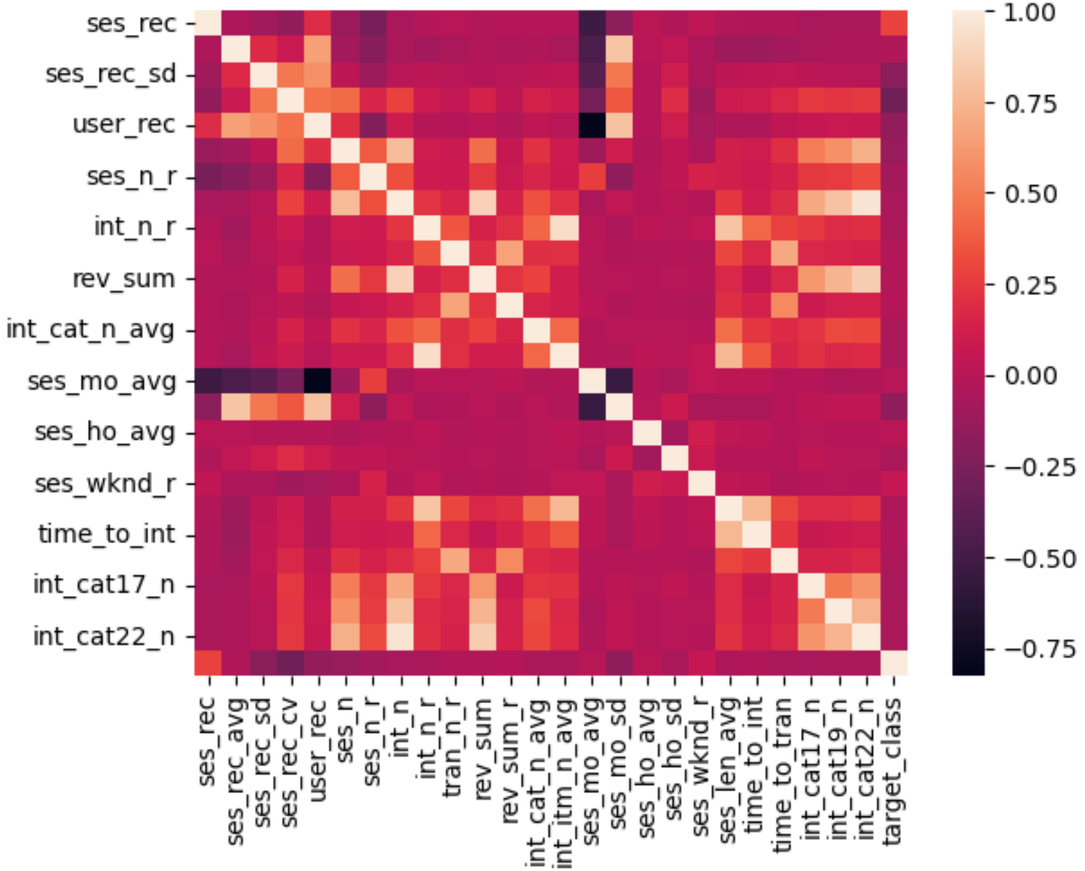
MRMR is a feature selection technique that strikes a balance between selecting features based on their relevance to the target variable and minimizing redundancy among features.

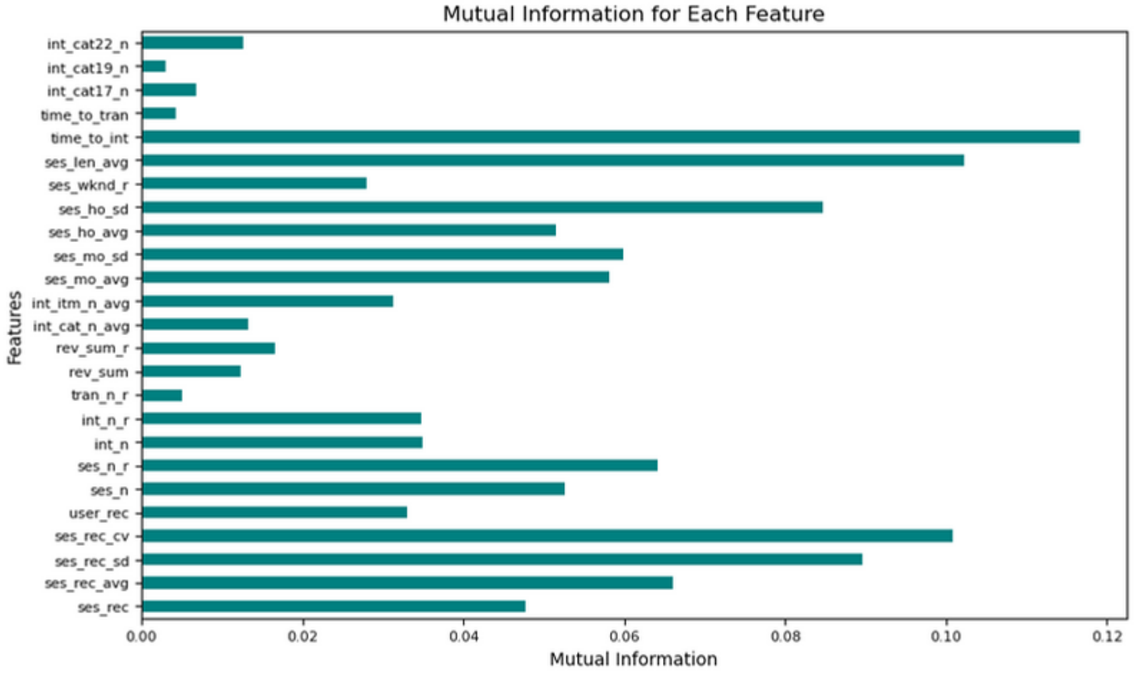


Feature Selection with SelectKBest

This method selects features based on mutual information, a metric quantifying the dependency between variables. Specifically, it identifies the top 25 features with the highest mutual information concerning the target variable

DATA AFTER SELECTION





ALGORITHMS

A. XGBoost

```python
import xgboost as xgb
from sklearn.model_selection import GridSearchCV

# Define the hyperparameter grid
param_grid = {
    'max_depth': [3, 5, 7],
    'learning_rate': [0.1, 0.01, 0.001],
    'subsample': [0.5, 0.7, 1]
}

# Create the XGBoost model object
xgb_model = xgb.XGBClassifier()

# Create the GridSearchCV object
grid_search = GridSearchCV(xgb_model, param_grid, cv=5, scoring='accuracy')

# Fit the GridSearchCV object to the training data
grid_search.fit(X_train, y_train)

# Print the best set of hyperparameters and the corresponding score
print("Best set of hyperparameters: ", grid_search.best_params_)
print("Best score: ", grid_search.best_score_)
```

```
Best set of hyperparameters:  {'learning_rate': 0.1, 'max_depth': 7, 'subsample': 0.
Best score:  0.7645312500000001
```

```
Model  Metrics:
              precision    recall  f1-score   support

           0       0.78      0.74      0.76      3996
           1       0.76      0.79      0.77      4004

    accuracy                           0.77      8000
   macro avg       0.77      0.77      0.77      8000
weighted avg       0.77      0.77      0.77      8000

Confusion Matrix:
 [[2970 1026]
 [ 830 3174]]
```

ALGORITHMS

B. Support Vector Machine

```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import classification_report, confusion_matrix



# Feature Scaling
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Creating the SVM model
svm_model = SVC(kernel='linear')

# Training the model
svm_model.fit(X_train, y_train)

# Making predictions
ys_pred = svm_model.predict(X_test)

# Evaluating the model
print(confusion_matrix(y_test, ys_pred))
print(classification_report(y_test, ys_pred))
```

```
[[2479 1517]
 [1081 2923]]
              precision    recall  f1-score   support

           0       0.70      0.62      0.66      3996
           1       0.66      0.73      0.69      4004

    accuracy                           0.68      8000
   macro avg       0.68      0.68      0.67      8000
weighted avg       0.68      0.68      0.67      8000
```

ALGORITHMS

C. Logistic Regression

```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.multiclass import OneVsRestClassifier

# Creating the Logistic Regression model
log_reg_model = OneVsRestClassifier(LogisticRegression())

# Training the model
log_reg_model.fit(X_train, y_train)

# Making predictions
yl_pred = log_reg_model.predict(X_test)

# Evaluating the model
print(confusion_matrix(y_test, yl_pred))
print(classification_report(y_test, yl_pred))
```

```
[[2495 1501]
 [1121 2883]]
              precision    recall  f1-score   support

           0       0.69      0.62      0.66      3996
           1       0.66      0.72      0.69      4004

    accuracy                           0.67      8000
   macro avg       0.67      0.67      0.67      8000
weighted avg       0.67      0.67      0.67      8000
```

ALGORITHMS

D. Random Forest

```python
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.datasets import make_classification
random_forest_model = RandomForestClassifier(n_estimators=100, random_state=42)
random_forest_model.fit(X_train, y_train)

# Make predictions using the trained model on the test set
y_pred_rf = random_forest_model.predict(X_test)

# Evaluate the performance of the Random Forest model
accuracy_rf = accuracy_score(y_test, y_pred_rf)
print(f"Accuracy of Random Forest Model: {accuracy_rf}")
print("\nClassification Report:\n", classification_report(y_test, y_pred_rf))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_rf))
```

```
Accuracy of Random Forest Model: 0.933375

Classification Report:
              precision    recall  f1-score   support

           0       0.90      0.97      0.94      3996
           1       0.97      0.90      0.93      4004

    accuracy                           0.93      8000
   macro avg       0.94      0.93      0.93      8000
weighted avg       0.94      0.93      0.93      8000

Confusion Matrix:
 [[3879  117]
 [ 416 3588]]
```

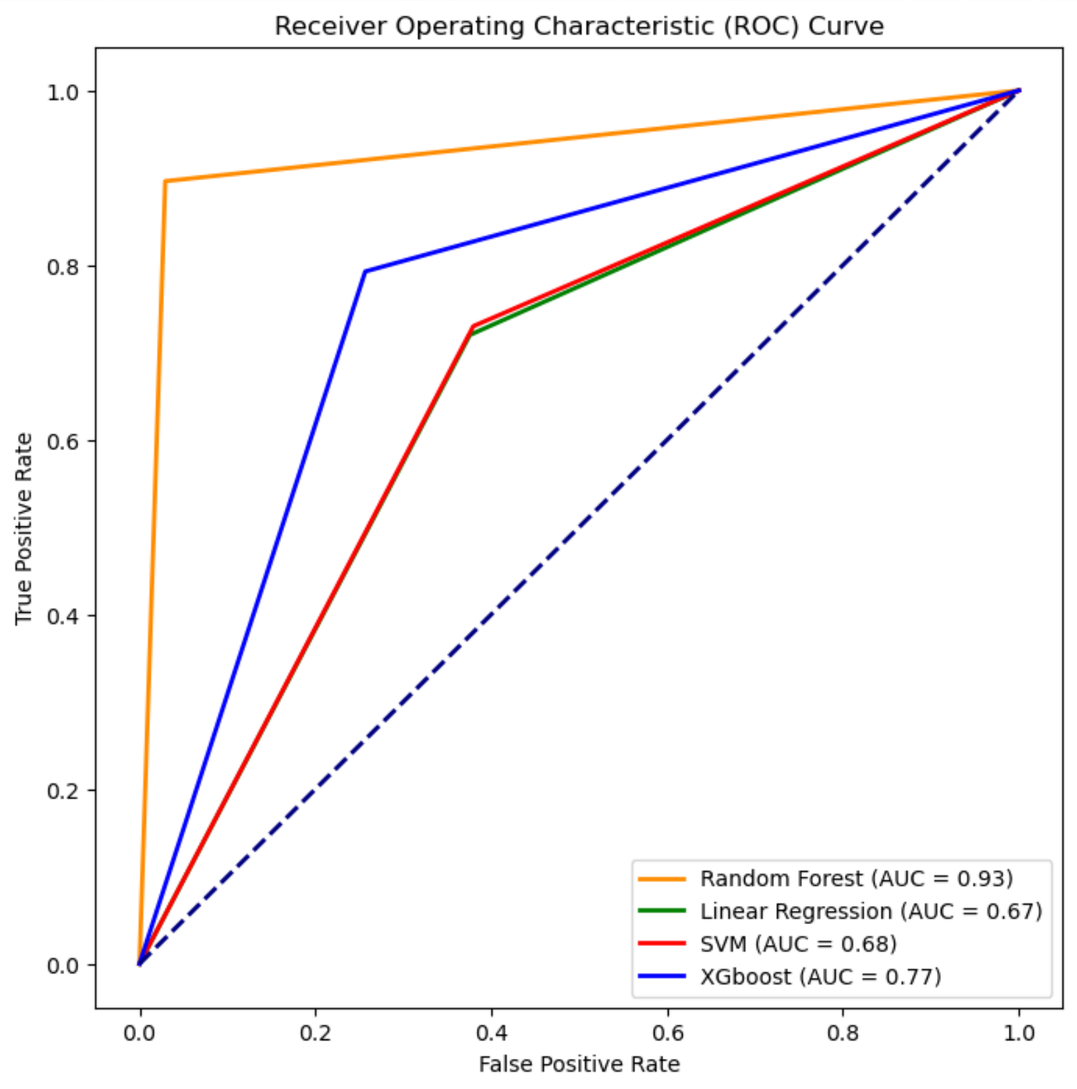
ENSEMBLE LEARNING

```python
from sklearn.ensemble import VotingClassifier
from sklearn.metrics import accuracy_score

# Assuming xgb_model, svm_model, and log_reg_model are your previously trained models

# Combine the models into a list of tuples
ensemble_models = [('xgboost', xgb_model), ('forest', random_forest_model), ('logistic', log_reg_model)]

# Create the voting classifier, using hard voting
voting_clf = VotingClassifier(estimators=ensemble_models, voting='hard')

# Fit the ensemble model
voting_clf.fit(X_train, y_train)

# Making predictions
y_pred_ensemble = voting_clf.predict(X_test)

# Evaluating the ensemble model
accuracy = accuracy_score(y_test, y_pred_ensemble)
print(f"Accuracy of Ensemble Model: {accuracy}")
```

```python
print(classification_report(y_test, y_pred_ensemble))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_ensemble))
```

```
              precision    recall  f1-score   support

           0       0.82      0.79      0.80      3996
           1       0.80      0.82      0.81      4004

    accuracy                           0.81      8000
   macro avg       0.81      0.81      0.81      8000
weighted avg       0.81      0.81      0.81      8000

Confusion Matrix:
 [[3154  842]
 [ 706 3298]]
```

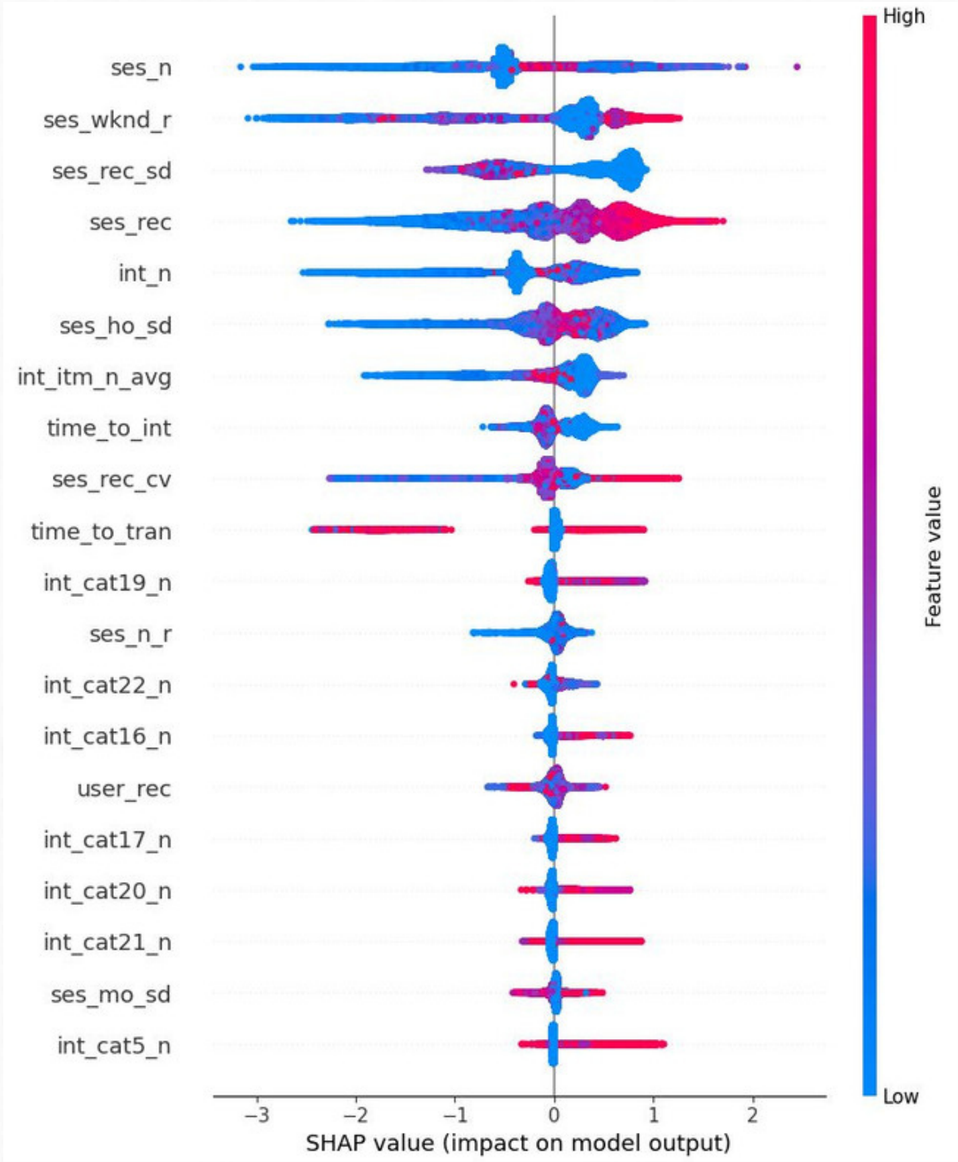
CONCLUSIONS



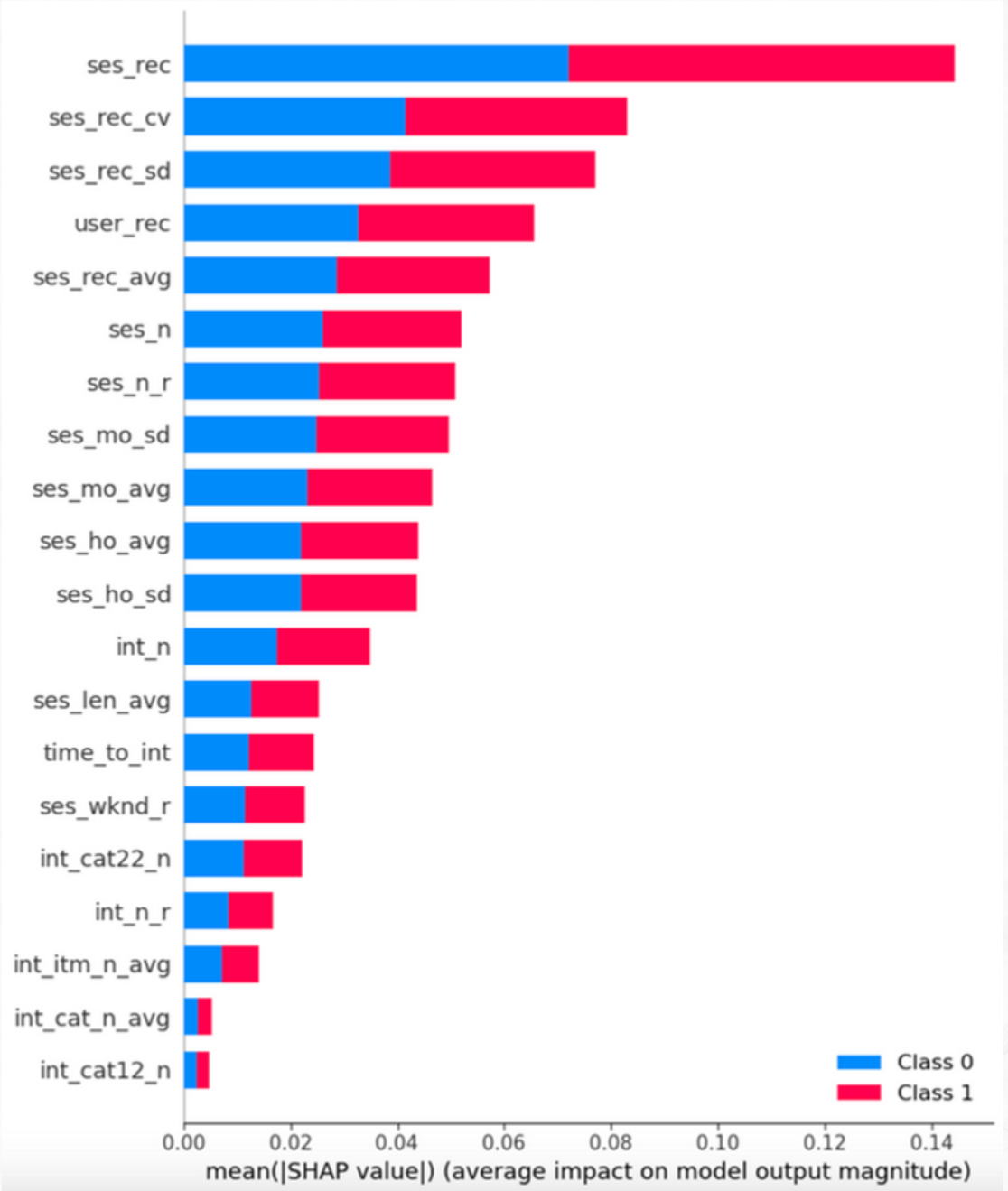
BEST MODEL

As you can see from the Roc Curve our best model is Radom Forest.

EXPLAINABLE AI ON XGBOOST



EXPLAINABLE AI ON RANDOM FOREST



THANK YOU

FOR YOUR ATTENTION