

Machine Learning

Task 1

Nathan Hefner

B. Describe the purpose of this data mining report by doing the following:

1. Propose one question relevant to a real-world organizational situation

My question, I propose, is what are the top factors for customers to churn? I will be answering this question using the Gradient Boost method.

2. Define one goal of the data analysis. Ensure that your goal is reasonable within the scope of the scenario and is represented in the available data.

My goal will be to find the top factors of what causes customers to churn and see how we can change them to minimize the chances of customers churning, therefore increasing retention.

C. Explain the reasons for your chosen classification method from part B1 by doing the following:

1. Explain how the classification method you chose analyzes the selected dataset.

Include expected outcomes.

To explain, we must first ensure the target variable is binary, which in our case is true because customers are either churned or not. The Gradient Boost process builds a series of decision trees where each new tree learns from the mistakes of previous trees, and the model will minimize the loss for binary classification by sequentially adding weak learners. A step-by-step process is as follows: the first tree makes initial predictions which are often inaccurate at the start, the model calculator how far these predictions are from the actual churn values, the next tree is then trained to predict those errors aka the residuals, this process will then repeat with each tree correcting the previous one, the final prediction is then calculated using the weighted sum of all trees.

I picked this method because it can handle both numeric and categorical data, through encoding, it can handle missing values, and it has built-in features to prevent overfitting.

The expected outcome would use the customer ID and give a percentage of their chance to churn.

2. List the packages or libraries you have chosen for Python or R, and justify how each item on the list supports the analysis.

Here are all the packages and libraries we will need to complete this task:

1. Pandas - Loads the dataset, allows Python to read the dataset, and helps clean the dataset if necessary.
2. NumPy - Used for efficient mathematical computations and supports xgboost and sklearn
3. XGBoost - Implements the Gradient Boosting algorithm for our classification method

4. Scikit-learn - Used to import train_test_split to split the data, standard scaler for normalization, and helps with model evaluation
5. Shap - Interprets singular features
6. Matplotlib & Seaborn - Visualizes plots or graphs created from a dataset

D. Perform data preparation for the chosen dataset by doing the following:

1. Describe one data preprocessing goal relevant to the classification method from part B1.

The first thing for preprocessing the data we need to do is ensure there are no empty or null cells, make sure there are no leading or trailing spaces, and drop all columns that are not relevant to us. The code below that is as seen below.

```
import pandas as pd
import numpy as np
import xgboost as xgb
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, roc_auc_score
import shap
import matplotlib.pyplot as plt
import seaborn as sns
```

```
file_path = r"C:\Users\Nathan\Documents\WGU\D603\Task 1\churn_clean.csv"
df = pd.read_csv(file_path)
```

```
print(df.isnull().sum())
```

CaseOrder	0
Customer_id	0
Interaction	0
UID	0
City	0
State	0
County	0
Zip	0
Lat	0
Lng	0
Population	0
Area	0
TimeZone	0
Job	0
Children	0
Age	0
Income	0
Marital	0
Gender	0
Churn	0
Outage_sec_perweek	0
Email	0
Contacts	0

```
df.columns = df.columns.str.strip()
```

```
df = df.drop(['CaseOrder', 'Customer_id', 'UID', 'Interaction',  
             'City', 'State', 'County', 'Zip', 'Lat', 'Lng', 'Email', 'TimeZone', 'Area'], axis=1)
```

Finally, the gradient boosting classification method can only operate with numeric data. The churn dataset will have numerous categorical variables that will be relevant to us. We will need to encode that data to keep the data consistent to numeric only. The code for that is seen below.

```
df = pd.get_dummies(df, columns=['Job', 'Marital', 'Gender', 'Churn', 'Techie', 'Contract', 'Port_modem',  
                                'Tablet', 'InternetService', 'Phone', 'Multiple', 'OnlineSecurity', 'OnlineBackup', 'DeviceProtection', 'TechSupport', 'StreamingTV',  
                                'StreamingMovies', 'PaperlessBilling', 'PaymentMethod'], drop_first=True)
```

I then exported this cleaned, processed dataset.

```
df.to_csv('churn_cleaned.csv', index=False)
```

2. Identify the initial dataset variables that you will use to perform the analysis for the classification question from part B1, and classify each variable as continuous or categorical.

After dropping the irrelevant data columns I separated the remaining ones into continuous or categorical below.

Categorical:

- Job
- Marital
- Gender
- Churn
- Techie
- Contract
- Port_modem
- Tablet
- InternetService
- Phone
- Multiple
- OnlineSecurity
- OnlineBackup
- DeviceProtection
- TechSupport
- StreamingTV
- StreamingMovies
- PaperlessBilling

- PaymentMethod

Continuous:

- Population
- Children
- Age
- Income
- Outage_sec_perweek
- Contacts
- Yearly_equip_failure
- Tenure
- MonthlyCharge
- Bandwidth_GB_Year
- Item1
- Item2
- Item3
- Item4
- Item5
- Item6
- Item7
- Item8

E. Perform the data analysis and report on the results by doing the following:

1. Split the data into training, validation, and test datasets

To split the datasets into the three subsets, we will need to utilize the library `sklearn.model_selection` and import `train_test_split`. We will first split the training for 20%, and from the validation, which is only 80% now, we will split the test by 25%. So in the end, the total percentages will be test - 20%, training - 20%, and validation 60%. The code is below.

```
train_val, test = train_test_split(df, test_size=0.2, random_state=42)

train, val = train_test_split(train_val, test_size=0.25, random_state=42)

train.to_csv('train.csv', index=False)
val.to_csv('validation.csv', index=False)
test.to_csv('test.csv', index=False)

print("Training, Validation, and Test CSV files have been successfully exported.")
```

Training, Validation, and Test CSV files have been successfully exported.

2. Create an initial model using the training dataset

Below is the code for creating a model using the initial training dataset.

```
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score, confusion_matrix

X_train = train.drop('Churn_Yes', axis=1)
y_train = train['Churn_Yes']

X_val = val.drop('Churn_Yes', axis=1)
y_val = val['Churn_Yes']

model = GradientBoostingClassifier(n_estimators=100, random_state=42)
model.fit(X_train, y_train)

y_pred = model.predict(X_val)
y_pred_prob = model.predict_proba(X_val)[:, 1]

accuracy = accuracy_score(y_val, y_pred)
precision = precision_score(y_val, y_pred)
recall = recall_score(y_val, y_pred)
f1 = f1_score(y_val, y_pred)
auc = roc_auc_score(y_val, y_pred_prob)
conf_matrix = confusion_matrix(y_val, y_pred)

print(f"Accuracy: {accuracy}")
print(f"Precision: {precision}")
print(f"Recall: {recall}")
print(f"F1 Score: {f1}")
print(f"AUC-ROC: {auc}")
print("Confusion Matrix:")
print(conf_matrix)

Accuracy: 0.8825
Precision: 0.8114754098360656
Recall: 0.7346938775510204
F1 Score: 0.7711781888997079
AUC-ROC: 0.9505497924389095
Confusion Matrix:
[[1369  92]
 [ 143 396]]
```

With an accuracy of 0.8825, we can see the model is predicting well, and the classification method we chose is also working well with our dataset.

3. Perform hyperparameter tuning on the validation dataset using k-fold cross validation to find the optimized model.

Below is my code for performing hyperparameter tuning on the validation dataset.

```
from sklearn.model_selection import GridSearchCV, StratifiedKFold
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import f1_score

param_grid = {
    'n_estimators': [100, 200, 500],
    'max_depth': [3, 4, 5],
    'learning_rate': [0.05, 0.1, 0.2],
    'subsample': [0.8, 1.0]
}

cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

model = GradientBoostingClassifier(random_state=42)

grid = GridSearchCV(model, param_grid, scoring='f1', cv=cv, n_jobs=-1)

grid.fit(X_train, y_train)

print("Best Params :", grid.best_params_)
print("Best F1 score during CV :", grid.best_score_)
```

```
Best Params : {'learning_rate': 0.2, 'max_depth': 3, 'n_estimators': 100, 'subsample': 1.0}
Best F1 score during CV : 0.7947797838729395
```

I chose four parameters for hyperparameter training being `n_estimators`, `max_depth`, `learning_rate`, and `subsample`.

1. `N_estimators` - This parameter controls the number of boosted trees. Each new tree focuses on the errors made by the previous ones. We gave it options of 100, 200, and 500 because if there are too few, the model may underfit, and if there are too many, the model overfits.
2. `Max_depth` - This parameter sets the maximum depth for each weak learner. A small depth makes the base classifiers weak, reducing variance, and a large depth makes them more powerful, which leads to overfitting. By trying 3, 4, and 5, it can find a good mark where the trees are strong enough to learn patterns but not overfit the data.
3. `Learning_rate` - This parameter is the scaling factor for the contribution of each weak classifier to the final prediction. Lower values slow down the algorithm by adding trees more carefully. By trialing 0.05, 0.1, and 0.2 to find the balance between speed and performance.
4. `Subsample` - This parameter is the fraction of the training set to sample when growing each new tree. If the value is less than 1, it adds randomness to help reduce variance and overfitting. So by trying 0.8 and 1.0, we see if adding randomness improves the model.

4. Use the optimized model identified in part E3 to make predictions using the test dataset

Below is my code and results from my optimized model.

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score, confusion_matrix

optimized_model = grid.best_estimator_

y_pred = optimized_model.predict(X_val)
y_pred_prob = optimized_model.predict_proba(X_val)[:, 1]

accuracy = accuracy_score(y_val, y_pred)
precision = precision_score(y_val, y_pred)
recall = recall_score(y_val, y_pred)
f1 = f1_score(y_val, y_pred)
auc = roc_auc_score(y_val, y_pred_prob)

print(f"Accuracy: {accuracy}")
print(f"Precision: {precision}")
print(f"Recall: {recall}")
print(f"F1 Score: {f1}")
print(f"AUC-ROC: {auc}")
print("Confusion Matrix:")
print(confusion_matrix(y_val, y_pred))
```

```
Accuracy: 0.884
Precision: 0.8063872255489022
Recall: 0.7495361781076066
F1 Score: 0.7769230769230769
AUC-ROC: 0.9489510196462381
Confusion Matrix:
[[1364  97]
 [ 135 404]]
```

F. Summarize your data analysis by doing the following:

1. Compare and discuss the metrics of accuracy, precision, recall, F1 score, and AUC-ROC from the use of the optimized model on the test dataset and the initial model on the training dataset to evaluate the performance of the optimized model.

A quick summary of my unoptimized model:

- Accuracy - 0.8825
- Precision - 0.8115
- Recall - 0.7347
- F1 - 7712
- AUC-ROC - 0.9505
- Confusion Matrix - $\begin{bmatrix} \frac{1369 (TN)}{143 (FN)} & \frac{92 (FP)}{396 (TP)} \end{bmatrix}$

And a quick summary of my optimized model

- Accuracy - 0.884
- Precision - 0.8064
- Recall - 0.7495
- F1 - 7769
- AUC-ROC - 0.9489
- Confusion Matrix - $\begin{bmatrix} \frac{1364 (TN)}{135 (FN)} & \frac{97 (FP)}{404 (TP)} \end{bmatrix}$

Let's go through this one at a time. For accuracy, we see that the optimized model improved by 0.015 percent when dealing with a massive dataset can be significant. Precision had a slight drop when predicting churn rates of 0.051. Recall increased by 0.148, which means it is able to identify more churners and miss fewer now. F1 score increased by 0.0057, which means the balance between precision and recall improved. AUC-ROC dropped slightly by 0.0016, which means there wasn't much change. Most importantly, the confusion matrix improved overall by having an increase of true positives (from 396 to 404, going up by 8) and the false negatives decreased (from 143 to 135, dropping by 8 as well).

Overall, the optimized model is performing better because it is able to identify more actual churners without losing much precision.

2. Discuss the results and implications of your classification analysis.

The results show that the model is very accurate when it comes to predicting whether a customer will churn or not. The implications of this analysis are that this model and type of classification are a reliable way of training the model, optimizing it, and then testing it, which will lead to accurate predictions and reliable results.

3. Discuss one limitation of your data analysis.

A limitation to this data analysis was simply not having enough data for concrete results. This will then cause generalization, which is caused by poor performance, a class imbalance, meaning the amount of not churned customers is more than churned, which means they are weighted differently and can cause problems. Because of this, the model can miss hidden features or trends to cause churn rates, not giving us the entire reason.

4. Recommend a course of action for the real-world organizational situation from part B1 based on your results and implications discussed in part F2.

A course of action I would take is to collect more data for the model to extrapolate from, and to make the churned versus not churned rates more even, so the model is not over-generalizing. This will help the model make better and more accurate predictions while also reducing the chances of missing hidden trends or features that would cause a customer to churn.

Sources

No sources were used except WGU Material