02_preprocessing

October 21, 2024

1 Preprocessing of the SyriaTel Customer Churn Dataset

[1]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import joblib

data_path = "../data/raw/telecom_churn_dataset.csv"
df = pd.read_csv(data_path)

df.head()
```

[1]:
```
  state  account length  area code phone number international plan  \
0    KS             128        415     382-4657                 no
1    OH             107        415     371-7191                 no
2    NJ             137        415     358-1921                 no
3    OH              84        408     375-9999                yes
4    OK              75        415     330-6626                yes

  voice mail plan  number vmail messages  total day minutes  total day calls  \
0             yes                     25              265.1              110
1             yes                     26              161.6              123
2              no                      0              243.4              114
3              no                      0              299.4               71
4              no                      0              166.7              113

   total day charge  …  total eve calls  total eve charge  \
0             45.07  …               99             16.78
1             27.47  …              103             16.62
2             41.38  …              110             10.30
3             50.90  …               88              5.26
4             28.34  …              122             12.61

   total night minutes  total night calls  total night charge  \
0                244.7                 91               11.01
1                254.4                103               11.45
2                162.6                104                7.32
```

```
3                196.9                 89                8.86
4                186.9                121                8.41

   total intl minutes  total intl calls  total intl charge  \
0                10.0                 3               2.70
1                13.7                 3               3.70
2                12.2                 5               3.29
3                 6.6                 7               1.78
4                10.1                 3               2.73

   customer service calls  churn
0                       1  False
1                       1  False
2                       0  False
3                       2  False
4                       3  False

[5 rows x 21 columns]
```

1.1 Formatting Column Names and Dropping Unnecessary Columns

Below, I am formatting the column names so that they are all following the same standard. I am also dropping `state` and `phone_number` as they are not relevant for predicting customer churn.

[2]:
```
df.columns = df.columns.str.strip().str.lower().str.replace(' ', '_')

print(df.columns)
```

```
Index(['state', 'account_length', 'area_code', 'phone_number',
       'international_plan', 'voice_mail_plan', 'number_vmail_messages',
       'total_day_minutes', 'total_day_calls', 'total_day_charge',
       'total_eve_minutes', 'total_eve_calls', 'total_eve_charge',
       'total_night_minutes', 'total_night_calls', 'total_night_charge',
       'total_intl_minutes', 'total_intl_calls', 'total_intl_charge',
       'customer_service_calls', 'churn'],
      dtype='object')
```

[3]:
```
df.drop(['phone_number', 'state'], axis=1, inplace=True)

df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3333 entries, 0 to 3332
Data columns (total 19 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   account_length          3333 non-null   int64
 1   area_code               3333 non-null   int64
 2   international_plan      3333 non-null   object
```

```
 3   voice_mail_plan         3333 non-null   object
 4   number_vmail_messages   3333 non-null   int64
 5   total_day_minutes       3333 non-null   float64
 6   total_day_calls         3333 non-null   int64
 7   total_day_charge        3333 non-null   float64
 8   total_eve_minutes       3333 non-null   float64
 9   total_eve_calls         3333 non-null   int64
 10  total_eve_charge        3333 non-null   float64
 11  total_night_minutes     3333 non-null   float64
 12  total_night_calls       3333 non-null   int64
 13  total_night_charge      3333 non-null   float64
 14  total_intl_minutes      3333 non-null   float64
 15  total_intl_calls        3333 non-null   int64
 16  total_intl_charge       3333 non-null   float64
 17  customer_service_calls  3333 non-null   int64
 18  churn                   3333 non-null   bool
dtypes: bool(1), float64(8), int64(8), object(2)
memory usage: 472.1+ KB
```

2 Encoded Categorical Variables

Below, I applied one-hot encoding to convert the categorical features (`international_plan` and `voice_mail_plan`) into numerical values, allowing them to be used in the model.

[4]:
```
# label encoding the binary categorical variables
df['international_plan'] = df['international_plan'].map({'yes': 1, 'no': 0})
df['voice_mail_plan'] = df['voice_mail_plan'].map({'yes': 1, 'no': 0})

# verifying the encoding
print(df[['international_plan', 'voice_mail_plan']].head())
```

```
   international_plan  voice_mail_plan
0                   0                1
1                   0                1
2                   0                0
3                   1                0
4                   1                0
```

[5]:
```
# verifying that the data types have changed
print(df.dtypes)
```

```
account_length              int64
area_code                   int64
international_plan          int64
voice_mail_plan             int64
number_vmail_messages       int64
total_day_minutes         float64
total_day_calls             int64
```

```
total_day_charge          float64
total_eve_minutes         float64
total_eve_calls             int64
total_eve_charge          float64
total_night_minutes       float64
total_night_calls           int64
total_night_charge        float64
total_intl_minutes        float64
total_intl_calls            int64
total_intl_charge         float64
customer_service_calls      int64
churn                        bool
dtype: object
```

3 Class Imbalance Note

It has been noted that there is class imbalance present, this will be handled during the model training phase (either through oversampling, undersampling, or class weights).

4 Feature Scaling and Data Splitting

Features were scaled using `StandardScaler` to ensure all variables contribute equally during model training. Additionally, the dataset was split into training and testing sets, with 80% of the data used for training and 20% for testing.

[6]:
```
# separating features (X) and target (y)
X = df.drop('churn', axis=1)
y = df['churn']

# scaling the features using StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

df_preprocessed = pd.concat([pd.DataFrame(X_scaled, columns=X.columns), y.
 ↪reset_index(drop=True)], axis=1)
joblib.dump(df_preprocessed, '../data/processed/preprocessed_data.pkl')
```

[6]:
```
['../data/processed/preprocessed_data.pkl']
```

[7]:
```
# splitting the data into training and testing sets (80/20 split)
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2,␣
 ↪random_state=42)
```

[8]:
```
# saving the split data
joblib.dump(X_train, '../data/processed/X_train.pkl')
joblib.dump(X_test, '../data/processed/X_test.pkl')
joblib.dump(y_train, '../data/processed/y_train.pkl')
```

```
joblib.dump(y_test, '../data/processed/y_test.pkl')

# saving the scaled dataframe
joblib.dump(X_scaled, '../data/processed/X_scaled.pkl')
```

[8]: ['../data/processed/X_scaled.pkl']

4.1 Conclusion of Preprocessing

In this notebook, I have: - Dropped unnecessary columns. - Encoded categorical variables. - Scaled the features. - Split the dataset into training and testing sets.

I will now proceed with model building and evaluation, which will be performed in a separate notebook.