churn_prediction_prepare_data

January 23, 2023

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import OneHotEncoder, LabelEncoder
from datetime import date, timedelta
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
import imblearn
from imblearn.over_sampling import SMOTE
from sklearn.metrics import precision_score, recall_score, f1_score,␣
↪roc_auc_score
from sklearn.metrics import confusion_matrix
from xgboost import XGBClassifier
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import accuracy_score
from sklearn.ensemble import RandomForestClassifier
import datetime
from datetime import date, timedelta
```

Data preparation and pre-processing

Functions

[2]:
```
### Functions

def load_data(path):
    return (pd.read_csv(path))

def prepare_data(df_transactions, df_players):

    df_transactions.columns = ["x"]
    df_transactions = df_transactions["x"].str.split(';', expand = True)
    df_transactions.columns = ["player_id", "transaction_date", "product",␣
↪"transaction_type", "amount", "count"]

    #change types
    df_transactions['player_id'] = df_transactions['player_id'].astype('int32')
```

```python
    df_transactions['transaction_date'] = df_transactions['transaction_date'].astype('datetime64')
    df_transactions['product'] = df_transactions['product'].astype('string')
    df_transactions['transaction_type'] = df_transactions['transaction_type'].astype('string')
    df_transactions['amount'] = df_transactions['amount'].astype('float32')
    df_transactions['count'] = df_transactions['count'].astype('int32')

    df_players.columns = ["player_id", "birth_date", "city", "registration_date", "registration_hour", "is_opt_out", "registration_terminal"]

    #change types
    df_players['birth_date'] = df_players['birth_date'].astype('datetime64')
    df_players['city'] = df_players['city'].astype('string')
    df_players['registration_date'] = df_players['registration_date'].astype('datetime64')
    df_players['registration_hour'] = df_players['registration_hour'].astype('float32')

    return df_transactions, df_players

def get_new_features(df):

    # Prepare for calculating new features later

    # Time since registration
    df['time_since_registration'] = (df['transaction_date'] - df['registration_date']).dt.total_seconds()

    # Frequency and monetary value
    df['prev_amount'] = df.groupby('player_id')['amount'].shift(1)
    df['prev_amount'].fillna(df['amount'], inplace = True)

    df['frequency'] = (df.groupby('player_id')['transaction_date'].cumcount() + 1)

    df['monetary_value'] = df.groupby('player_id')['prev_amount'].cumsum()

    df.drop(['prev_amount'], axis = 1, inplace = True)


    return df

def clean_data(df):
```

```
    len_before_clean = len(df)

    # For some rows transaction_date is before registration_date which is not␣
↪possible
    df = df.drop(df[df['transaction_date'] < df['registration_date']].index)

    # We have some null values (very small amount so best option is to delete)
    df = df.dropna()

    len_after_clean = len(df)

    print("Removed ", (1 - len_after_clean / len_before_clean) * 100, "% of␣
↪data.", sep = '')

    return df
```

Load dataset and prepare to calculate new features

[3]:
```
df_transactions = load_data("dataset/zadatak-lite.csv")
df_players = load_data("dataset/igraci.csv")

df_transactions, df_players = prepare_data(df_transactions, df_players)

df = pd.merge(df_transactions, df_players, on = 'player_id', how = 'left')

df = clean_data(df)

df.sort_values(by = 'transaction_date', inplace = True)
```

```
Removed 5.001921881014326% of data.
```

[4]:
```
df = get_new_features(df)
```

[5]:
```
print(df['transaction_date'].min(), df['transaction_date'].max(), sep = '\n')
```

```
2022-04-01 00:00:00
2022-12-31 00:00:00
```

[6]:
```
start_date = date(2022, 5, 1)
end_date = date(2022, 11, 30)
date_range = [start_date + timedelta(days=x) for x in range((end_date -␣
↪start_date).days + 1)]

new_df = pd.DataFrame(columns = ['player_id', 'date', 'bo_count',␣
↪'casino_count', 'pp_count', 'sport_count', 'vb_count',
        'vdr_count', 'frequency', 'monetary_value', 'profit', 'deposit',
        'last_active', 'player_age', 'time_since_registration', 'is_opt_out',
```

```
        'churn'])

for d in date_range:
    #print(d)
    d = pd.to_datetime(d)

    last_30_days_df = df[(df['transaction_date'] >= d - timedelta(days = 30)) &␣
 ↪(df['transaction_date'] < d)]

    # Count by product
    # Group by player_id, product
    product_count_by_player = last_30_days_df.
 ↪groupby(['player_id','product'])['count'].sum().reset_index()
    product_count_by_player = product_count_by_player.pivot(index =␣
 ↪'player_id', columns = 'product', values = 'count')
    # Fillna with 0
    product_count_by_player.fillna(0, inplace=True)
    product_count_by_player.rename(columns = {'Sport': 'sport_count','Casino':
 ↪'casino_count',
                                              'PaymentProvider':
 ↪'pp_count','BusinessOwner':'bo_count',
                                              'VirtualBingo':'vb_count',␣
 ↪'VirtualDogRace': 'vdr_count'},
                                               inplace = True)

    # Frequency
    frequency_by_player = last_30_days_df.groupby(['player_id'])['frequency'].
 ↪max() - last_30_days_df.groupby(['player_id'])['frequency'].min()

    frequency_by_player = frequency_by_player.to_frame()
    frequency_by_player = frequency_by_player.rename(columns={0: "frequency"})


    # Monetary value
    monetary_by_player = last_30_days_df.
 ↪groupby(['player_id'])['monetary_value'].max() - last_30_days_df.
 ↪groupby(['player_id'])['monetary_value'].min()

    monetary_by_player = monetary_by_player.to_frame()
    monetary_by_player = monetary_by_player.rename(columns={0:␣
 ↪"monetary_value"})

    # Profit, Deposit
    transaction_amount_by_player = last_30_days_df.groupby(['player_id',␣
 ↪'transaction_type'])['amount'].sum().reset_index()
```

```python
    transaction_amount_by_player = transaction_amount_by_player.pivot(index = 'player_id', columns = 'transaction_type', values = 'amount')
    transaction_amount_by_player.fillna(0, inplace = True)
    transaction_amount_by_player.rename(columns = {'Bonus': 'bonus', 'TicketWin': 'ticketwin','TicketPayin':'payin','Deposit':'deposit','Withdrawal':'withdrawal'}, inplace = True)
    profit = transaction_amount_by_player['ticketwin'] + transaction_amount_by_player['bonus'] - transaction_amount_by_player['payin']
    transaction_amount_by_player.insert(0, 'profit', profit)
    profit_by_player = transaction_amount_by_player.drop(columns = ['bonus', 'payin',
                                        'ticketwin', 'withdrawal', 'DepositCancel', 'TicketPayinCancel', 'TicketWinCancel'])

    # Last active
    last_active_by_player = (d - last_30_days_df.groupby(['player_id'])['transaction_date'].max())

    last_active_by_player = last_active_by_player.to_frame()
    last_active_by_player = last_active_by_player.rename(columns={'transaction_date': "last_active"})

    # Player age
    player_age = d - last_30_days_df.groupby(['player_id'])['birth_date'].max()

    player_age = player_age.to_frame()
    player_age = player_age.rename(columns={'birth_date': "player_age"})

    #Time since registration
    time_since_registration = d - last_30_days_df.groupby(['player_id'])['registration_date'].max()

    time_since_registration = time_since_registration.to_frame()
    time_since_registration = time_since_registration.rename(columns={'registration_date': "time_since_registration"})

    # Is opt out
    is_opt_out = last_30_days_df.groupby(['player_id'])['is_opt_out'].max()

    is_opt_out = is_opt_out.to_frame()
    is_opt_out = is_opt_out.rename(columns={0: "is_opt_out"})

    # Merge all features in one dataframe
    concat_df = pd.concat([product_count_by_player, frequency_by_player, monetary_by_player,
                           profit_by_player, last_active_by_player, player_age,
```

```
                        time_since_registration, is_opt_out], axis=1)

    # Fix for player_id column
    concat_df['player_id'] = concat_df.index

    # Churn
    concat_df['churn'] = concat_df['player_id'].isin(df[(df['transaction_date']␣
↪> d) &
                                (df['transaction_date'] < d + timedelta(days =␣
↪30))]['player_id']).astype(int)

    concat_df['churn'] = concat_df['churn'].replace({0:1, 1:0})

    # Date
    concat_df['date'] = d

    # Append to complete dataframe

    new_df['frequency'] = new_df['frequency'].astype('int')
    new_df['is_opt_out'] = new_df['is_opt_out'].astype('int')
    new_df['player_id'] = new_df['player_id'].astype('int')
    new_df['churn'] = new_df['churn'].astype('int')

    new_df = pd.concat([new_df, concat_df], ignore_index = True)


new_df = new_df.drop('WithdrawalCancel', axis = 1)
new_df.head()
```

[6]:

```
   player_id       date  bo_count  casino_count  pp_count  sport_count  \
0          2 2022-05-01       1.0           0.0       1.0         13.0
1          4 2022-05-01      11.0       16992.0      79.0         25.0
2          5 2022-05-01       2.0       19796.0      43.0          0.0
3          8 2022-05-01       0.0           0.0       1.0          8.0
4         11 2022-05-01       1.0           0.0       0.0          0.0

   vb_count  vdr_count  frequency  monetary_value      profit     deposit  \
0       0.0        0.0          2       38.009998   -1.809998    1.810000
1    6153.0        0.0        167     2495.419922 -258.329956  257.429993
2       0.0        0.0         28     7263.290039 -411.770020  592.309998
3       0.0        0.0          7        4.910000   -1.630000    0.900000
4       0.0        0.0          0        0.000000    4.520000    0.000000

  last_active player_age time_since_registration  is_opt_out  churn
0     22 days 10168 days               2777 days           0      0
1      2 days 18747 days               2777 days           0      0
2      6 days 15087 days               2777 days           0      0
```

```
3       1 days 18619 days                2776 days            0       0
4      23 days  9963 days                2776 days            0       1
```

Save csv file

[7]:
```
# Save dataframe ready for model
tmp_df = new_df
tmp_df.to_csv('dataset/new_feature_dataset.csv')
```

churn_prediction_model

January 23, 2023

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import OneHotEncoder, LabelEncoder
from datetime import date, timedelta
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
import imblearn
from imblearn.over_sampling import SMOTE
from sklearn.metrics import precision_score, recall_score, f1_score,␣
↪roc_auc_score
from sklearn.metrics import confusion_matrix
from xgboost import XGBClassifier
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import accuracy_score
from sklearn.ensemble import RandomForestClassifier
import datetime
from datetime import date, timedelta
```

Load dataset

[14]:
```
df = pd.read_csv('dataset/new_feature_dataset.csv')
df.drop(['Unnamed: 0'], axis=1, inplace=True)
df['last_active'] = df['last_active'].apply(lambda x: int(x.split(" ")[0]))
df['player_age'] = df['player_age'].apply(lambda x: int(x.split(" ")[0]))
df['time_since_registration'] = df['time_since_registration'].apply(lambda x:␣
↪int(x.split(" ")[0]))
df.head()
```

[14]:
```
   player_id        date  bo_count  casino_count  pp_count  sport_count  \
0          2  2022-05-01       1.0           0.0       1.0         13.0
1          4  2022-05-01      11.0       16992.0      79.0         25.0
2          5  2022-05-01       2.0       19796.0      43.0          0.0
3          8  2022-05-01       0.0           0.0       1.0          8.0
4         11  2022-05-01       1.0           0.0       0.0          0.0

   vb_count  vdr_count  frequency  monetary_value      profit  deposit  \
```

```
0     0.0      0.0        2          38.01    -1.809998    1.81
1  6153.0      0.0      167        2495.42  -258.329960  257.43
2     0.0      0.0       28        7263.29  -411.770020  592.31
3     0.0      0.0        7           4.91    -1.630000    0.90
4     0.0      0.0        0           0.00     4.520000    0.00

   last_active  player_age  time_since_registration  is_opt_out  churn
0           22       10168                     2777           0      0
1            2       18747                     2777           0      0
2            6       15087                     2777           0      0
3            1       18619                     2776           0      0
4           23        9963                     2776           0      1
```

Split the data into training and test sets

[3]:
```python
def split_df_by_date(df, date_col, date):
    df_before = df[df[date_col] < date]
    df_after = df[df[date_col] >= date]
    return df_before, df_after
```

[4]:
```python
df_train, df_test = split_df_by_date(df, 'date', '2022-10-30')
```

[5]:
```python
#X_train = df_train[['bo_count', 'casino_count', 'pp_count', 'sport_count',␣
 ↪'vb_count', 'vdr_count',
#                    'frequency', 'monetary_value', 'profit', 'deposit',␣
 ↪'last_active', 'player_age',
#                    'time_since_registration', 'is_opt_out']]
#X_test = df_test[['bo_count', 'casino_count', 'pp_count', 'sport_count',␣
 ↪'vb_count', 'vdr_count',
#                    'frequency', 'monetary_value', 'profit', 'deposit',␣
 ↪'last_active', 'player_age',
#                    'time_since_registration', 'is_opt_out']]

X_train = df_train[['frequency', 'monetary_value', 'last_active']]
X_test = df_test[['frequency', 'monetary_value', 'last_active']]

y_train = df_train[['churn']].values.ravel()
y_test = df_test[['churn']].values.ravel()

smote = SMOTE(sampling_strategy = "minority")
```

Logistic Regression

[6]:
```python
clf = LogisticRegression(penalty = None, random_state = 42, max_iter = 10000)
clf.fit(X_train, y_train)
```

[6]:
```
LogisticRegression(max_iter=10000, penalty=None, random_state=42)
```

Optimal threshold

Ovdje smo optimizirali tako da zahtijevamo da barem 85% pravih churnera naš model detektira kao churnere, a u isto vrijeme minimiziramo našu metriku koja nam govori koliko smo ukupno igrača detektirali kao churnere. Recimo da šaljemo bonuse svim igračima koje model detektira kao churnere. Ovisno o tome koliko profita nam donosi ispravno detektiranje churnera i koliki je gubitak ako non-churnera detektiramo kao churnera ili churnera kao non-churnera mijenjali bismo fiksni postotak koji je ovdje 85%.

[7]:
```
#thresholds = np.arange(0, 1.05, 0.01)
thresholds = np.arange(0.14, 0.16, 0.0001)

metrics = []

for threshold in thresholds:

    y_pred = (clf.predict_proba(X_test)[:,1] >= threshold)
    tn, fp, fn, tp = confusion_matrix(y_test, y_pred).ravel()

    custom_metric = (tp + fp) / (tp + fp + tn + fn)
    rec = tp / (tp + fn)

    if(rec > 0.85):
        metrics.append(custom_metric)
    else:
        metrics.append(2.0)

best_threshold = thresholds[np.argmin(metrics)]
best_custom_metric = np.min(metrics)

y_pred = (clf.predict_proba(X_test)[:,1] >= best_threshold)

print("Best threshold:", best_threshold)
print("Best custom metric:", best_custom_metric)
```

```
Best threshold: 0.1445999999999995
Best custom metric: 0.3305623744699844
```

Optimize company profit

[8]:
```
# Send bonus to churner
profit_tp = 5.0
# Send bonus to non_churner
profit_fp = -1.0
# Did not send bonus to churner
profit_fn = -5.0

#thresholds = np.arange(0, 1.05, 0.01)
thresholds = np.arange(0.11, 0.13, 0.0001)
```

```
profits = []

for threshold in thresholds:

    y_pred = (clf.predict_proba(X_test)[:,1] >= threshold)
    tn, fp, fn, tp = confusion_matrix(y_test, y_pred).ravel()

    profit = profit_tp * tp + profit_fp * fp + profit_fn * fn

    profits.append(profit)

best_threshold = thresholds[np.argmax(profits)]
best_profit = np.max(profits)

y_pred = (clf.predict_proba(X_test)[:,1] >= best_threshold)

print("Best threshold:", best_threshold)
print("Best profits:", best_profit)
```

```
Best threshold: 0.12540000000000046
Best profits: 55781.0
```

Calculate important metrics

[9]:
```
# calculate accuracy
accuracy = clf.score(X_test, y_test)
print("Accuracy: {:.2f}%".format(accuracy*100))

# calculate precision
precision = precision_score(y_test, y_pred)
print("Precision: ", precision)

# calculate recall
recall = recall_score(y_test, y_pred)
print("Recall: ", recall)

# calculate f1-score
f1 = f1_score(y_test, y_pred)
print("F1-Score: ", f1)

# calculate AUC-ROC
auc_roc = roc_auc_score(y_test, y_pred)
print("AUC-ROC: ", auc_roc)
```

```
Accuracy: 88.04%
Precision:  0.2912517918716826
Recall:  0.8754257765873824
```

```
F1-Score:  0.43708618897763685
AUC-ROC:  0.7927765680528369
```

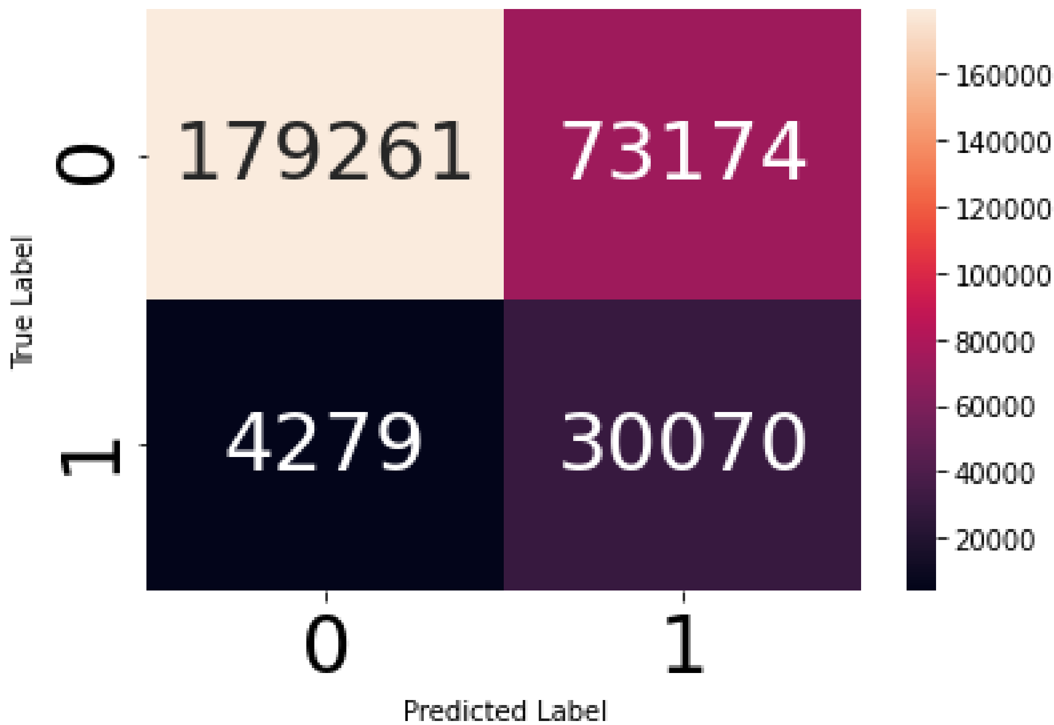
Confusion matrix

[10]:
```
cm = confusion_matrix(y_test, y_pred)

# plot the confusion matrix
sns.heatmap(cm, annot=True, fmt="d", annot_kws={"size": 30})

plt.xticks(fontsize = 30)
plt.yticks(fontsize = 30)


plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```



Correlation matrix

[11]:
```
# Adjust the size of the figure
plt.figure(figsize = (20, 10))

# Create the heatmap with larger annotations
```

```
sns.heatmap(df.corr(), annot = True, fmt = ".2f", cmap = "YlGnBu", linewidths=.
↪5, annot_kws = {"size": 20})

# Increase the font size of the labels on the x and y axes
plt.xticks(fontsize = 20)
plt.yticks(fontsize = 20)

# Show the plot
plt.show()
```

```
/tmp/ipykernel_40049/2151628175.py:5: FutureWarning: The default value of
numeric_only in DataFrame.corr is deprecated. In a future version, it will
default to False. Select only valid columns or specify the value of numeric_only
to silence this warning.
  sns.heatmap(df.corr(), annot = True, fmt = ".2f", cmap = "YlGnBu",
linewidths=.5, annot_kws = {"size": 20})
```