Case Analysis: “Pentathlon Cross-sell/Upsell Modeling”

Professor Song Yao
Olin Business School

Customer Analytics

This case is about customizing e-mail messages to consumers

PENTATHLON NPTB SETUP

- Use how consumers reacted to last promotional e-mail blast to predict which message works best for which consumer
- E-mail blast randomly allocates messages to consumers
  --> Perfect setup to estimate the effect of different messages
- 600,000 customers
  - This is a fairly large dataset, reducing the need to cross-validation when using simple logistic and linear models
  - Caveat: For more sophisticated modeling approaches (e.g., neural net, random forest, or even logistic/linear models with regularization), we normally still need cross-validation.

This case is about customizing e-mail messages to consumers

PENTATHLON NPTB VARIABLES AND SAMPLES

- **Dependent Variables:**
  - **buyer** – Did the customer click on the e-mail and complete a purchase within two days of receiving the e-mail (if yes, buyer=1, 0 otherwise)?
  - **total_os**– Total order size (in Euro) conditional on the customer having purchased (buyer=1). Measures spending for all products, not just for department that sent the message.
- **Independent Variables:**
  - Demographics: **age**, **female**, **income**, **education**, **children**
  - **Frequency** of purchase over last year for each of 7 departments
- **Samples:**
  - training=1 -> 80% of the data (random seed set at 42)
  - training=0 -> 20% of the data

The case deals with 4 related analytics problems

PENTATHLON Cross-selling and Upselling PROBLEM

1. Predict for each customer the **probability of purchasing** and **order size** after having been sent an e-mail with each of the 7 possible messages.
   --> "The Analysis," questions 1-2
2. Use predictions of purchase probability and order size to customize message for each customer.
   --> "The Analysis," questions 3-5
3. Evaluate the **incremental revenue** from customization
   --> "The Analysis," question 6-8
4. Evaluate the new **e-mail policy** proposal

Step 1 and Step 2, the probability of purchase and the order size of purchase

- E[Profit] = Prob(buy) * (Profit if customer buys)
  = <u>Prob(buy) * (Order size if customer buys)</u> * (1-COGS)
- **Step 1:** Interact message with demographics and frequencies to obtain *<u>individual-specific</u>* purchase probability for each message
- **Step 2:** Interact message with demographics and frequencies to obtain *<u>individual-specific</u>* order size (if customer buys) for each message

First take a look at the variable types (and their summary statistics)

String/Character variables need to be treated as "categorical"

```
# Display information about the dataframe
print("\nDataframe Info:")
pentathlon.info()

# Display summary statistics of the dataframe
print("\nSummary Statistics:")
print(pentathlon.describe())

# Check for missing values
print("\nMissing Values:")
print(pentathlon.isnull().sum())
```

```
Data types of each column:
custid               int64
buyer                int64
total_os             int64
message             object
age                 object
female               int64
income               int64
education            int64
children           float64
freq_endurance       int64
freq_strength        int64
freq_water           int64
freq_team            int64
freq_backcountry     int64
freq_winter          int64
freq_racquet         int64
```

We estimate the probability of purchase including the message interaction

ESTIMATING LOGIT MODEL AND CHECK OVERFITTING

```
# Split into training and test sets (80/20)
from sklearn.model_selection import train_test_split
train_data, test_data = train_test_split(pentathlon, test_size=0.2, random_state=42)
```

```
# Create a formula string with interactions
formula_interactions = "buyer ~ C(message) * (C(age) + female + income + education + children + \
    freq_endurance + freq_strength + freq_water + freq_team + freq_backcountry + freq_winter + freq_racquet)"


# Train model on training data
logit_interactions = smf.logit(formula=formula_interactions, data=train_data)
logit_interactions_results = logit_interactions.fit()

# Print summary of results
print(logit_interactions_results.summary())

# Get predictions on train and test set
y_pred_proba_train_interactions = logit_interactions_results.predict(train_data)
y_pred_proba_test_interactions = logit_interactions_results.predict(test_data)

# Calculate and print AUC score for both train and test set
auc_score_train_interactions = roc_auc_score(train_data['buyer'], y_pred_proba_train_interactions)
auc_score_test_interactions = roc_auc_score(test_data['buyer'], y_pred_proba_test_interactions)
print(f"\nTrain Set AUC Score: {auc_score_train_interactions:.3f}")
print(f"\nTest Set AUC Score: {auc_score_test_interactions:.3f}")
```

```
Train Set AUC Score: 0.792

Test Set AUC Score: 0.788
```


We estimate the order size including the message interaction

ESTIMATING LINEAR MODEL AND CHECK OVERFITTING

```
# Filter training data where buyer=1
buyers_train = train_data[train_data['buyer'] == 1]
buyers_test = test_data[test_data['buyer'] == 1]

# Create formula for linear regression using same features as above
formula_linear = "total_os ~ C(message) * (C(age) + female + income + education + children + \
    freq_endurance + freq_strength + freq_water + freq_team + freq_backcountry + freq_winter + freq_racquet)"

# Train linear regression model for total_os
os_model = smf.ols(formula=formula_linear, data=buyers_train)
os_results = os_model.fit()

# Print summary of results
print("\nLinear Regression Results for Buyers:")
print(os_results.summary())

# Get predictions on train and test set
y_pred_train_linear = os_results.predict(buyers_train)
y_pred_test_linear = os_results.predict(buyers_test)

# Calculate and print MSE and MAE scores for both train and test set of buyers
from sklearn.metrics import mean_squared_error, mean_absolute_error
mse_train = mean_squared_error(buyers_train['total_os'], y_pred_train_linear)
mae_train = mean_absolute_error(buyers_train['total_os'], y_pred_train_linear)
mse_test = mean_squared_error(buyers_test['total_os'], y_pred_test_linear)
mae_test = mean_absolute_error(buyers_test['total_os'], y_pred_test_linear)
print(f"\nTrain Set MSE Score: {mse_train:.3f}")
print(f"\nTrain Set MAE Score: {mae_train:.3f}")
print(f"\nTest Set MSE Score: {mse_test:.3f}")
print(f"\nTest Set MAE Score: {mae_test:.3f}")
```

```
Train Set MSE Score: 3578.566

Train Set MAE Score: 37.968

Test Set MSE Score: 3885.803

Test Set MAE Score: 39.024
```

Predict each message's purchase probability for each customer

```
# Combine train and test data for full dataset analysis
full_data = pd.concat([train_data, test_data])

# Create a DataFrame to store probabilities for each message type
message_probs = pd.DataFrame()

# Get all unique message types from the data
message_types = full_data['message'].unique()

# For each message type, predict probability
for message in message_types:
    # Create temporary DataFrame with current message
    temp_data = full_data.copy()
    temp_data['message'] = message

    # Predict probabilities using the interaction model
    probs = logit_interactions_results.predict(temp_data)
    message_probs[message] = probs

# Find the message that gives highest probability for each customer
target_message = message_probs.idxmax(axis=1) #this picks the message label
target_message_prob = message_probs.max(axis=1) #this picks the probability

# Add the target message to the original DataFrame
full_data['target_message'] = target_message
full_data['target_message_prob'] = target_message_prob
```


Use max purchase probability to customize message for each customer

```
# Print distribution of target messages
print("\nDistribution of Target Messages (%):")
print((full_data['target_message'].value_counts(normalize=True)*100).round(2))

# Print the average purchase probs of different messages
print('The average purchase probs of different messages are:')
print(message_probs.mean())

# Calculate the mean predicted probability for each target message
mean_probs_target_message = \
    full_data.groupby('target_message')['target_message_prob'].mean()

print("\nMean Predicted Purchase Probability by Target Message:")
print(mean_probs_target_message.sort_values(ascending=False))
```

```
Distribution of Target Messages (%):
target_message
endurance      44.04
strength       26.61
water          17.13
racquet         7.27
backcountry     2.45
team            2.10
winter          0.40
Name: proportion, dtype: float64
```

```
The average purchase probs of different messages are:
strength       0.029971
backcountry    0.027685
endurance      0.030744
water          0.029534
racquet        0.027126
winter         0.028331
team           0.027673
dtype: float64

Mean Predicted Purchase Probability by Target Message:
target_message
winter         0.130289
team           0.088943
backcountry    0.057820
racquet        0.054952
water          0.031414
strength       0.030789
endurance      0.025897
Name: target_message_prob, dtype: float64
```

See the differences in probabilities, e.g., winter and team. Why?

Predict each message's expected order size for each customer

```python
# Create DataFrames to store predictions for each message type
os_predictions = pd.DataFrame()
expected_os = pd.DataFrame()

# For each message type, predict total_os and calculate expected value
for message in message_types:
    # Create temporary DataFrame with current message
    temp_data = full_data.copy()
    temp_data['message'] = message

    # Predict total_os and purchase probability using respective models
    os_pred = os_results.predict(temp_data)
    prob_pred = logit_interactions_results.predict(temp_data)

    # Calculate expected total_os (probability * total_os)
    expected_os[message] = prob_pred * os_pred

# Find the message that gives highest expected total_os for each customer
target_message_os = expected_os.idxmax(axis=1)
target_message_os_number = expected_os.max(axis=1)

# Add the target message based on expected total_os to the original DataFrame
full_data['target_message_os'] = target_message_os
full_data['target_message_os_number'] = target_message_os_number
```


Use max order size to customize message for each customer

```python
# Print distribution of target messages based on expected total_os
print("\nDistribution of Target Messages based on Expected Total OS (%):")
print((full_data['target_message_os'].value_counts(normalize=True)*100).round(2))

# Print the average expected total_os for each message
print('The average expected total_os for each message are:')
print(expected_os.mean())

# Calculate the mean expected total_os for each target message
mean_expected_os = full_data.groupby('target_message_os')['target_message_os_number'].mean()

print("\nMean Expected Total OS by Target Message:")
print(mean_expected_os.sort_values(ascending=False))
```

```
Distribution of Target Messages based on Expected Total OS (%):
target_message_os
water          44.93
backcountry    16.13
winter         15.52
endurance      13.85
racquet         6.48
strength        1.99
team            1.10
```

```
The average expected total_os for each message are:
strength       1.637886
backcountry    1.731318
endurance      1.698230
water          1.824755
racquet        1.543906
winter         1.745024
team           1.569413
dtype: float64

Mean Expected Total OS by Target Message:
target_message_os
team           9.081201
strength       4.654955
racquet        3.646869
endurance      2.368574
winter         1.832932
backcountry    1.807924
water          1.533388
Name: target_message_os_number, dtype: float64
```

Again, see the differences in order size, e.g., team.

Customers receive very different messages when we maximize order size instead of purchase probability

Based on order size

```
Distribution of Target Messages
target_message_os
water          44.93
backcountry    16.13
winter         15.52
endurance      13.85
racquet         6.48
strength        1.99
team            1.10
```

Based on purchase prob

```
Distribution of Target Messages
target_message
endurance      44.04
strength       26.61
water          17.13
racquet         7.27
backcountry     2.45
team            2.10
winter          0.40
```


Next, we evaluate the incremental profit (order size) from the customization

PROFITABILITY RESULTS

```
# Calculate total expected OS across all customers using the expected OS based approach
avg_expected_os = expected_os.max(axis=1).mean()

print("\nAverage Expected OS across all customers using expected OS based approach:")
print(f"{avg_expected_os:,.2f}")

# Calculate actual total OS from original data
actual_avg_os = full_data['total_os'].mean()

print("\nActual Average OS from original data:")
print(f"{actual_avg_os:,.2f}")

# Calculate percentage increase from actual to expected OS
percentage_increase = ((avg_expected_os - actual_avg_os) / actual_avg_os * 100)

print("\nPercentage increase from actual to expected OS:")
print(f"{percentage_increase:.2f}%")
```

```
Average Expected OS across all customers using expected OS based approach:
2.02

Actual Average OS from original data:
1.67

Percentage increase from actual to expected OS:
20.90%
```

Targeting vs. Random Profit improvement: 1.67 -> 2.02 ~ 20.9%

Fourth, we evaluate the new e-mail policy proposal: Why do top messages only have 70% (instead of 100%)?

E-MAIL POLICY PROPOSAL

A. Each customer's email frequency will be one email per week. The weekly featured department of a customer's email will be determined on a weekly basis using the preceding week's data.

B. We assign customer emails to departments using the following procedure:

   a. For each customer, the analytics team forecasts the messages that yield the highest and the second highest expected order sizes among the seven possible messages.

   b. Between the two messages from the previous step (i.e., with highest and second highest expected order sizes), the top message receives 70% chance to be featured in that customer's weekly email, and the second message receives 30% chance to be featured.