Chapter 4: Simple Logistic Regression Modeling

Created	@September 20, 2024 12:17 AM
Class	Introduction to Regression with statsmodels in Python

Logistic Regression for Binary Response Variables

Key Details

- Used when the response variable is binary (0 or 1)
- Example dataset: Customer churn in a European financial services company
- Logistic regression is a type of generalized linear model
- Predictions follow an S-shaped logistic curve
- Ensures predictions stay between 0 and 1 (valid probabilities)

Dataset Overview

- 400 rows, each representing a customer
- Response variable: has_churned (0 or 1)
- Explanatory variables:
    1. Time since first service purchase (relationship length)
    2. Time since last service purchase (recency of activity)

- Time variables standardized (contain negative values)

Linear Regression Limitations

Issues with Linear Models for Binary Data

- Predictions can be fractional (between 0 and 1)
- Can predict negative probabilities or probabilities greater than 1
- Doesn't accurately represent the binary nature of the data

```
# Linear regression example
linear_model = smf.ols('has_churned ~ time_since_last_purchas
e', data=churn_data).fit()
intercept, slope = linear_model.params

# Plotting linear regression
fig, ax = plt.subplots()
ax.scatter(churn_data['time_since_last_purchase'], churn_data
['has_churned'])
plt.axline(xy1=(0, intercept), slope=slope)
ax.set_xlabel('Time Since Last Purchase')
ax.set_ylabel('Probability of Churn')
plt.show()
```

Logistic Regression

Advantages

- Predictions always between 0 and 1
- S-shaped curve better represents binary outcomes

Implementation

```
from statsmodels.formula.api import logit
```

```
# Fitting logistic regression
logistic_model = logit('has_churned ~ time_since_last_purchas
e', data=churn_data).fit()

# Plotting logistic regression
fig, ax = plt.subplots()
sns.regplot(x='time_since_last_purchase', y='has_churned', da
ta=churn_data, logistic=True)
ax.set_xlabel('Time Since Last Purchase')
ax.set_ylabel('Probability of Churn')
plt.show()
```

Interpretation

- Small time since last purchase: probability of churning close to 0
- Large time since last purchase: probability of churning close to 1
- Customers who recently made purchases are less likely to churn

Comparison: Linear vs. Logistic Regression

```
# Plotting both models
fig, ax = plt.subplots()
ax.scatter(churn_data['time_since_last_purchase'], churn_data
['has_churned'])
plt.axline(xy1=(0, linear_intercept), slope=linear_slope, col
or='black', label='Linear')
sns.regplot(x='time_since_last_purchase', y='has_churned', da
ta=churn_data, logistic=True, color='blue', label='Logistic')
ax.set_xlabel('Time Since Last Purchase')
ax.set_ylabel('Probability of Churn')
ax.legend()
plt.show()
```

Key Takeaways

1. Logistic regression is appropriate for binary response variables
2. It ensures predictions are valid probabilities (between 0 and 1)
3. The S-shaped curve better represents the relationship in binary outcome data
4. Interpretation focuses on probability of the outcome occurring
5. In the churn example, recent customer activity correlates with lower churn probability

Making Predictions with Logistic Regression Models

Key Details

- Logistic regression predicts probabilities of binary outcomes
- Various ways to express predictions: probabilities, most likely outcomes, odds ratios, and log-odds ratios
- Each expression method has different benefits and interpretations

Making Predictions

Basic Probability Predictions

```
# Create DataFrame with explanatory variable values
prediction_data = pd.DataFrame({'time_since_last_purchase': n
p.linspace(churn_data['time_since_last_purchase'].min(), chur
n_data['time_since_last_purchase'].max(), 100)})

# Add predictions
prediction_data['churn_probability'] = logistic_model.predict
(prediction_data)

# Plotting predictions
fig, ax = plt.subplots()
```

```
ax.scatter(churn_data['time_since_last_purchase'], churn_data
['has_churned'])
ax.scatter(prediction_data['time_since_last_purchase'], predi
ction_data['churn_probability'], color='red')
ax.set_xlabel('Time Since Last Purchase')
ax.set_ylabel('Probability of Churn')
plt.show()
```

Most Likely Outcome

- Simplest prediction: probability < 0.5 (won't churn), probability > 0.5 (will churn)
- Easy to understand but lacks precision

```
prediction_data['most_likely_outcome'] = np.round(prediction_
data['churn_probability'])

# Plotting most likely outcome
fig, ax = plt.subplots()
ax.scatter(prediction_data['time_since_last_purchase'], predi
ction_data['most_likely_outcome'])
ax.set_xlabel('Time Since Last Purchase')
ax.set_ylabel('Most Likely Outcome (0: No Churn, 1: Churn)')
plt.show()
```

Odds Ratio

Definition and Calculation

- Odds ratio = probability of event / (1 - probability of event)
- Example: Probability of 0.25 equals odds of "3 to 1 against"

```
prediction_data['odds_ratio'] = prediction_data['churn_probab
ility'] / (1 - prediction_data['churn_probability'])
```

```
# Plotting odds ratio
fig, ax = plt.subplots()
sns.lineplot(x='time_since_last_purchase', y='odds_ratio', data=prediction_data)
plt.axhline(y=1, linestyle=':', color='red')
ax.set_xlabel('Time Since Last Purchase')
ax.set_ylabel('Odds Ratio of Churn')
plt.show()
```

Interpretation

- Odds ratio < 1: Churning less likely than not churning
- Odds ratio > 1: Churning more likely than not churning
- Odds ratio = 1: Churning and not churning equally likely

Log-Odds Ratio (Logit)

Properties

- Log-odds ratio changes linearly with the explanatory variable
- Also known as the logit

```
prediction_data['log_odds_ratio'] = np.log(prediction_data['odds_ratio'])

# Plotting log-odds ratio
fig, ax = plt.subplots()
sns.lineplot(x='time_since_last_purchase', y='log_odds_ratio', data=prediction_data)
ax.set_xlabel('Time Since Last Purchase')
ax.set_ylabel('Log-Odds Ratio of Churn')
ax.set_yscale('log')
plt.show()
```

Comparison of Prediction Methods

```
# Display all calculated values
print(prediction_data[['time_since_last_purchase', 'churn_pro
bability', 'most_likely_outcome', 'odds_ratio', 'log_odds_rat
io']])
```

Pros and Cons

1. Most Likely Outcome
   - Pro: Easiest to understand (yes/no)
   - Con: Lacks precision
2. Probabilities and Odds Ratios
   - Pro: Fairly easy to understand for data-literate audience
   - Con: Non-linear predictions make it hard to reason about changes in explanatory variables
3. Log-Odds Ratio (Logit)
   - Pro: Linear relationship with explanatory variables makes it easy to reason about changes
   - Con: Difficult to interpret for individual values

Key Takeaways

1. Logistic regression can predict probabilities, most likely outcomes, odds ratios, and log-odds ratios
2. Each prediction method has its own benefits and drawbacks in terms of interpretation and precision
3. The choice of prediction method depends on the audience and the specific needs of the analysis
4. Log-odds ratios (logit) provide a linear relationship with explanatory variables, making it useful for understanding variable effects

5. Visualizing different prediction methods can provide insights into the model's behavior and the relationship between variables

Assessing Logistic Regression Model Performance

Key Details

- Confusion matrices are used to evaluate logistic regression models
- Performance metrics include accuracy, sensitivity, and specificity
- Visualization tools like mosaic plots can help interpret confusion matrices

Confusion Matrix

Definition

- A table that categorizes predictions based on whether they match the actual values
- Four possible outcomes: True Negative, True Positive, False Negative, False Positive

Calculating Confusion Matrix

```
# Get actual responses
actual = churn_data['has_churned']

# Get predicted responses
predicted_probs = logistic_model.predict(churn_data)
predicted = np.round(predicted_probs)

# Create confusion matrix
confusion_df = pd.DataFrame({'actual': actual, 'predicted': p
redicted})
confusion_matrix = confusion_df.value_counts().unstack()
```

```
# Alternatively, use pred_table method
confusion_matrix_auto = logistic_model.pred_table()
```

Visualizing Confusion Matrix

```
from statsmodels.graphics.mosaicplot import mosaic

mosaic(confusion_df, ['actual', 'predicted'])
plt.show()
```

Performance Metrics

1. Accuracy

- Definition: Proportion of correct predictions
- Formula: (True Negatives + True Positives) / Total Observations
- Higher accuracy is better

```
accuracy = (confusion_matrix[0, 0] + confusion_matrix[1, 1])
/ confusion_matrix.sum()
```

2. Sensitivity (True Positive Rate)

- Definition: Proportion of actual positives correctly identified
- Formula: True Positives / (True Positives + False Negatives)
- Higher sensitivity is better

```
sensitivity = confusion_matrix[1, 1] / (confusion_matrix[1,
1] + confusion_matrix[1, 0])
```

3. Specificity (True Negative Rate)

- Definition: Proportion of actual negatives correctly identified
- Formula: True Negatives / (True Negatives + False Positives)
- Higher specificity is better

```
specificity = confusion_matrix[0, 0] / (confusion_matrix[0,
0] + confusion_matrix[0, 1])
```

Interpreting Results

Example Confusion Matrix

```
Actual | Predicted
-------+-----------
       | No Churn | Churn
-------+---------+-------
No Churn|   141   |  59
Churn   |   111   |  89
```

Interpretation

- True Negatives: 141 (correctly predicted no churn)
- True Positives: 89 (correctly predicted churn)
- False Positives: 59 (incorrectly predicted churn)
- False Negatives: 111 (incorrectly predicted no churn)

Performance Metrics Calculation

```
accuracy = (141 + 89) / (141 + 89 + 59 + 111)  # 0.575 or 57.
5%
sensitivity = 89 / (89 + 111)  # 0.445 or 44.5%
specificity = 141 / (141 + 59)  # 0.705 or 70.5%
```

Key Takeaways

1. Confusion matrices provide a comprehensive view of model performance for binary classification problems
2. Accuracy, sensitivity, and specificity offer different perspectives on model performance
3. There's often a trade-off between sensitivity and specificity
4. Visualizing the confusion matrix (e.g., using mosaic plots) can help in interpreting results
5. The choice of which metric to prioritize depends on the specific problem and the costs associated with different types of errors