Activity_Course 5 Waze project lab

June 6, 2024

1 Waze Project

Course 5 - Regression analysis: Simplify complex data relationships

Your team is more than halfway through their user churn project. Earlier, you completed a project proposal, used Python to explore and analyze Waze's user data, created data visualizations, and conducted a hypothesis test. Now, leadership wants your team to build a regression model to predict user churn based on a variety of variables.

You check your inbox and discover a new email from Ursula Sayo, Waze's Operations Manager. Ursula asks your team about the details of the regression model. You also notice two follow-up emails from your supervisor, May Santner. The first email is a response to Ursula, and says that the team will build a binomial logistic regression model. In her second email, May asks you to help build the model and prepare an executive summary to share your results.

A notebook was structured and prepared to help you in this project. Please complete the following questions and prepare an executive summary.

2 Course 5 End-of-course project: Regression modeling

In this activity, you will build a binomial logistic regression model. As you have learned, logistic regression helps you estimate the probability of an outcome. For data science professionals, this is a useful skill because it allows you to consider more than one variable against the variable you're measuring against. This opens the door for much more thorough and flexible analysis to be completed.

The purpose of this project is to demostrate knowledge of exploratory data analysis (EDA) and a binomial logistic regression model.

The goal is to build a binomial logistic regression model and evaluate the model's performance.

This activity has three parts:

Part 1: EDA & Checking Model Assumptions * What are some purposes of EDA before constructing a binomial logistic regression model?

Part 2: Model Building and Evaluation * What resources do you find yourself using as you complete this stage?

Part 3: Interpreting Model Results

- What key insights emerged from your model(s)?
- What business recommendations do you propose based on the models built?

Follow the instructions and answer the question below to complete the activity. Then, you will complete an executive summary using the questions listed on the PACE Strategy Document.

Be sure to complete this activity before moving on. The next course item will provide you with a completed exemplar to compare to your own work.

3 Build a regression model

4 PACE stages

Throughout these project notebooks, you'll see references to the problem-solving framework PACE. The following notebook components are labeled with the respective PACE stage: Plan, Analyze, Construct, and Execute.

4.1 PACE: Plan

Consider the questions in your PACE Strategy Document to reflect on the Plan stage.

4.1.1 Task 1. Imports and data loading

Import the data and packages that you've learned are needed for building logistic regression models.

[ ]:
```
# Packages for numerics + dataframes
### YOUR CODE HERE ###

# Packages for visualization
### YOUR CODE HERE ###

# Packages for Logistic Regression & Confusion Matrix
### YOUR CODE HERE ###
```

Import the dataset.

Note: As shown in this cell, the dataset has been automatically loaded in for you. You do not need to download the .csv file, or provide more code, in order to access the dataset and proceed with this lab. Please continue with this activity by completing the following instructions.

[ ]:
```
# Load the dataset by running this cell

df = pd.read_csv('waze_dataset.csv')
```

4.2 PACE: Analyze

Consider the questions in your PACE Strategy Document to reflect on the Analyze stage.

In this stage, consider the following question:

- What are some purposes of EDA before constructing a binomial logistic regression model?

==> ENTER YOUR RESPONSE HERE

4.2.1 Task 2a. Explore data with EDA

Analyze and discover data, looking for correlations, missing data, potential outliers, and/or duplicates.

Start with `.shape` and `info()`.

[ ]:
```
### YOUR CODE HERE ###
```

Question: Are there any missing values in your data?

==> ENTER YOUR RESPONSE HERE

Use `.head()`.

[ ]:
```
### YOUR CODE HERE ###
```

Use `.drop()` to remove the ID column since we don't need this information for your analysis.

[ ]:
```
### YOUR CODE HERE ###
```

Now, check the class balance of the dependent (target) variable, `label`.

[ ]:
```
### YOUR CODE HERE ###
```

Call `.describe()` on the data.

[ ]:
```
### YOUR CODE HERE ###
```

Question: Are there any variables that could potentially have outliers just by assessing at the quartile values, standard deviation, and max values?

==> ENTER YOUR RESPONSE HERE

4.2.2 Task 2b. Create features

Create features that may be of interest to the stakeholder and/or that are needed to address the business scenario/problem.

`km_per_driving_day` You know from earlier EDA that churn rate correlates with distance driven per driving day in the last month. It might be helpful to engineer a feature that captures this information.

1. Create a new column in `df` called `km_per_driving_day`, which represents the mean distance driven per driving day for each user.
2. Call the `describe()` method on the new column.

```
[ ]: # 1. Create `km_per_driving_day` column
     ### YOUR CODE HERE ###

     # 2. Call `describe()` on the new column
     ### YOUR CODE HERE ###
```

Note that some values are infinite. This is the result of there being values of zero in the `driving_days` column. Pandas imputes a value of infinity in the corresponding rows of the new column because division by zero is undefined.

1. Convert these values from infinity to zero. You can use `np.inf` to refer to a value of infinity.
2. Call `describe()` on the `km_per_driving_day` column to verify that it worked.

```
[ ]: # 1. Convert infinite values to zero
     ### YOUR CODE HERE ###

     # 2. Confirm that it worked
     ### YOUR CODE HERE ###
```

`professional_driver` Create a new, binary feature called `professional_driver` that is a 1 for users who had 60 or more drives **and** drove on 15+ days in the last month.

Note: The objective is to create a new feature that separates professional drivers from other drivers. In this scenario, domain knowledge and intuition are used to determine these deciding thresholds, but ultimately they are arbitrary.

To create this column, use the `np.where()` function. This function accepts as arguments: 1. A condition 2. What to return when the condition is true 3. What to return when the condition is false

```
Example:
x = [1, 2, 3]
x = np.where(x > 2, 100, 0)
x
array([  0,   0, 100])
```

```
[ ]: # Create `professional_driver` column
     ### YOUR CODE HERE ###
```

Perform a quick inspection of the new variable.

1. Check the count of professional drivers and non-professionals
2. Within each class (professional and non-professional) calculate the churn rate

[ ]:
```
# 1. Check count of professionals and non-professionals
### YOUR CODE HERE ###

# 2. Check in-class churn rate
### YOUR CODE HERE ###
```

The churn rate for professional drivers is 7.6%, while the churn rate for non-professionals is 19.9%. This seems like it could add predictive signal to the model.

4.3 PACE: Construct

After analysis and deriving variables with close relationships, it is time to begin constructing the model.

Consider the questions in your PACE Strategy Document to reflect on the Construct stage.

In this stage, consider the following question:

- Why did you select the X variables you did?

==> ENTER YOUR RESPONSE HERE

4.3.1 Task 3a. Preparing variables

Call `info()` on the dataframe to check the data type of the `label` variable and to verify if there are any missing values.

[ ]:
```
### YOUR CODE HERE ###
```

Because you know from previous EDA that there is no evidence of a non-random cause of the 700 missing values in the `label` column, and because these observations comprise less than 5% of the data, use the `dropna()` method to drop the rows that are missing this data.

[ ]:
```
# Drop rows with missing data in `label` column
### YOUR CODE HERE ###
```

Impute outliers You rarely want to drop outliers, and generally will not do so unless there is a clear reason for it (e.g., typographic errors).

At times outliers can be changed to the **median, mean, 95th percentile, etc.**

Previously, you determined that seven of the variables had clear signs of containing outliers:

- `sessions`
- `drives`
- `total_sessions`

- `total_navigations_fav1`
- `total_navigations_fav2`
- `driven_km_drives`
- `duration_minutes_drives`

For this analysis, impute the outlying values for these columns. Calculate the **95th percentile** of each column and change to this value any value in the column that exceeds it.

[ ]:
```
# Impute outliers
### YOUR CODE HERE ###
```

Call `describe()`.

[ ]:
```
### YOUR CODE HERE ###
```

Encode categorical variables Change the data type of the `label` column to be binary. This change is needed to train a logistic regression model.

Assign a `0` for all `retained` users.

Assign a `1` for all `churned` users.

Save this variable as `label2` as to not overwrite the original `label` variable.

Note: There are many ways to do this. Consider using `np.where()` as you did earlier in this notebook.

[ ]:
```
# Create binary `label2` column
### YOUR CODE HERE ###
```

4.3.2 Task 3b. Determine whether assumptions have been met

The following are the assumptions for logistic regression:

- Independent observations (This refers to how the data was collected.)
- No extreme outliers
- Little to no multicollinearity among X predictors
- Linear relationship between X and the **logit** of y

For the first assumption, you can assume that observations are independent for this project.

The second assumption has already been addressed.

The last assumption will be verified after modeling.

Note: In practice, modeling assumptions are often violated, and depending on the specifics of your use case and the severity of the violation, it might not affect your model much at all or it will result in a failed model.

Collinearity Check the correlation among predictor variables. First, generate a correlation matrix.

[ ]:
```
# Generate a correlation matrix
### YOUR CODE HERE ###
```

Now, plot a correlation heatmap.

[ ]:
```
# Plot correlation heatmap
### YOUR CODE HERE ###
```

If there are predictor variables that have a Pearson correlation coefficient value greater than the **absolute value of 0.7**, these variables are strongly multicollinear. Therefore, only one of these variables should be used in your model.

Note: 0.7 is an arbitrary threshold. Some industries may use 0.6, 0.8, etc.

Question: Which variables are multicollinear with each other?

==> ENTER YOUR RESPONSE HERE

4.3.3 Task 3c. Create dummies (if necessary)

If you have selected `device` as an X variable, you will need to create dummy variables since this variable is categorical.

In cases with many categorical variables, you can use pandas built-in `pd.get_dummies()`, or you can use scikit-learn's `OneHotEncoder()` function.

Note: Variables with many categories should only be dummied if absolutely necessary. Each category will result in a coefficient for your model which can lead to overfitting.

Because this dataset only has one remaining categorical feature (`device`), it's not necessary to use one of these special functions. You can just implement the transformation directly.

Create a new, binary column called `device2` that encodes user devices as follows:

- `Android` -> `0`
- `iPhone` -> `1`

[ ]:
```
# Create new `device2` variable
### YOUR CODE HERE ###
```

4.3.4 Task 3d. Model building

Assign predictor variables and target To build your model you need to determine what X variables you want to include in your model to predict your target—`label2`.

Drop the following variables and assign the results to `X`:

- `label` (this is the target)
- `label2` (this is the target)

- `device` (this is the non-binary-encoded categorical variable)
- `sessions` (this had high multicollinearity)
- `driving_days` (this had high multicollinearity)

Note: Notice that `sessions` and `driving_days` were selected to be dropped, rather than `drives` and `activity_days`. The reason for this is that the features that were kept for modeling had slightly stronger correlations with the target variable than the features that were dropped.

[ ]:
```
# Isolate predictor variables
### YOUR CODE HERE ###
```

Now, isolate the dependent (target) variable. Assign it to a variable called `y`.

[ ]:
```
# Isolate target variable
### YOUR CODE HERE ###
```

Split the data Use scikit-learn's `train_test_split()` function to perform a train/test split on your data using the X and y variables you assigned above.

Note 1: It is important to do a train test to obtain accurate predictions. You always want to fit your model on your training set and evaluate your model on your test set to avoid data leakage.

Note 2: Because the target class is imbalanced (82% retained vs. 18% churned), you want to make sure that you don't get an unlucky split that over- or under-represents the frequency of the minority class. Set the function's `stratify` parameter to `y` to ensure that the minority class appears in both train and test sets in the same proportion that it does in the overall dataset.

[ ]:
```
# Perform the train-test split
### YOUR CODE HERE ###
```

[ ]:
```
# Use .head()
### YOUR CODE HERE ###
```

Use scikit-learn to instantiate a logistic regression model. Add the argument `penalty = None`.

It is important to add `penalty = None` since your predictors are unscaled.

Refer to scikit-learn's logistic regression documentation for more information.

Fit the model on `X_train` and `y_train`.

[ ]:
```
### YOUR CODE HERE ###
```

Call the `.coef_` attribute on the model to get the coefficients of each variable. The coefficients are in order of how the variables are listed in the dataset. Remember that the coefficients represent the change in the **log odds** of the target variable for **every one unit increase in X**.

If you want, create a series whose index is the column names and whose values are the coefficients in `model.coef_`.

[ ]:
```
### YOUR CODE HERE ###
```

Call the model's `intercept_` attribute to get the intercept of the model.

[ ]:
```
### YOUR CODE HERE ###
```

Check final assumption Verify the linear relationship between X and the estimated log odds (known as logits) by making a regplot.

Call the model's `predict_proba()` method to generate the probability of response for each sample in the training data. (The training data is the argument to the method.) Assign the result to a variable called `training_probabilities`. This results in a 2-D array where each row represents a user in `X_train`. The first column is the probability of the user not churning, and the second column is the probability of the user churning.

[ ]:
```
# Get the predicted probabilities of the training data
### YOUR CODE HERE ###
```

In logistic regression, the relationship between a predictor variable and the dependent variable does not need to be linear, however, the log-odds (a.k.a., logit) of the dependent variable with respect to the predictor variable should be linear. Here is the formula for calculating log-odds, where p is the probability of response:

$$logit(p) = ln(\frac{p}{1-p})$$

1. Create a dataframe called `logit_data` that is a copy of `df`.
2. Create a new column called `logit` in the `logit_data` dataframe. The data in this column should represent the logit for each user.

[ ]:
```
# 1. Copy the `X_train` dataframe and assign to `logit_data`
### YOUR CODE HERE ###

# 2. Create a new `logit` column in the `logit_data` df
### YOUR CODE HERE ###
```

Plot a regplot where the x-axis represents an independent variable and the y-axis represents the log-odds of the predicted probabilities.

In an exhaustive analysis, this would be plotted for each continuous or discrete predictor variable. Here we show only `driving_days`.

[ ]:
```
# Plot regplot of `activity_days` log-odds
### YOUR CODE HERE ###
```

4.4 PACE: Execute

Consider the questions in your PACE Strategy Document to reflect on the Execute stage.

4.4.1 Task 4a. Results and evaluation

If the logistic assumptions are met, the model results can be appropriately interpreted.

Use the code block below to make predictions on the test data.

[ ]:
```
# Generate predictions on X_test
### YOUR CODE HERE ###
```

Now, use the `score()` method on the model with `X_test` and `y_test` as its two arguments. The default score in scikit-learn is **accuracy**. What is the accuracy of your model?

Consider: Is accuracy the best metric to use to evaluate this model?

[ ]:
```
# Score the model (accuracy) on the test data
### YOUR CODE HERE ###
```

4.4.2 Task 4b. Show results with a confusion matrix

Use the `confusion_matrix` function to obtain a confusion matrix. Use `y_test` and `y_preds` as arguments.

[ ]:
```
### YOUR CODE HERE ###
```

Next, use the `ConfusionMatrixDisplay()` function to display the confusion matrix from the above cell, passing the confusion matrix you just created as its argument.

[ ]:
```
### YOUR CODE HERE ###
```

You can use the confusion matrix to compute precision and recall manually. You can also use scikit-learn's `classification_report()` function to generate a table from `y_test` and `y_preds`.

[ ]:
```
# Calculate precision manually
### YOUR CODE HERE ###
```

[ ]:
```
# Calculate recall manually
### YOUR CODE HERE ###
```

[ ]:
```
# Create a classification report
### YOUR CODE HERE ###
```

Note: The model has decent precision but very low recall, which means that it makes a lot of false negative predictions and fails to capture users who will churn.

4.4.3 BONUS

Generate a bar graph of the model's coefficients for a visual representation of the importance of the model's features.

[ ]:
```
# Create a list of (column_name, coefficient) tuples
### YOUR CODE HERE ###

# Sort the list by coefficient value
### YOUR CODE HERE ###
```

[ ]:
```
# Plot the feature importances
### YOUR CODE HERE ###
```

4.4.4 Task 4c. Conclusion

Now that you've built your regression model, the next step is to share your findings with the Waze leadership team. Consider the following questions as you prepare to write your executive summary. Think about key points you may want to share with the team, and what information is most relevant to the user churn project.

Questions:

1. What variable most influenced the model's prediction? How? Was this surprising?
2. Were there any variables that you expected to be stronger predictors than they were?
3. Why might a variable you thought to be important not be important in the model?
4. Would you recommend that Waze use this model? Why or why not?
5. What could you do to improve this model?
6. What additional features would you like to have to help improve the model?

==> ENTER YOUR RESPONSES TO QUESTIONS 1-6 HERE

Congratulations! You've completed this lab. However, you may not notice a green check mark next to this item on Coursera's platform. Please continue your progress regardless of the check mark. Just click on the "save" icon at the top of this notebook to ensure your work has been logged.