Introduction

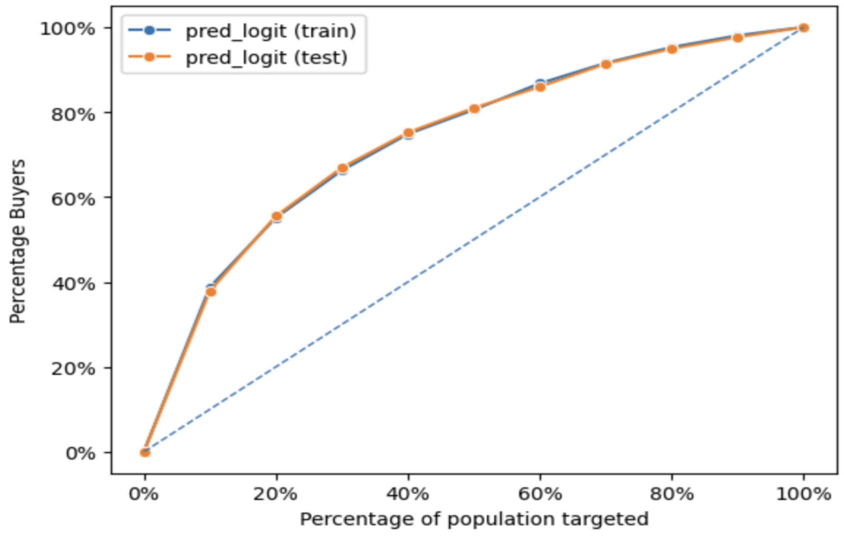
Intuit launched a QuickBooks Upgrade Campaign to encourage small businesses to transition to QuickBooks Version 3.0. The campaign began with Wave-1, where 801,821 businesses received a direct mail upgrade offer, with some responding and others not. The objective for Wave-2 is to identify and target the most likely responders among the non-responding businesses to maximize conversion rates and profitability.

Given the mailing cost of $1.41 per business and revenue of $60 per upgrade, an optimized strategy is essential to minimize costs while maximizing return on investment (ROI). Additionally, Wave-2 is expected to experience a 50% drop in response rates compared to Wave-1, requiring a more selective targeting approach

Feature Selection

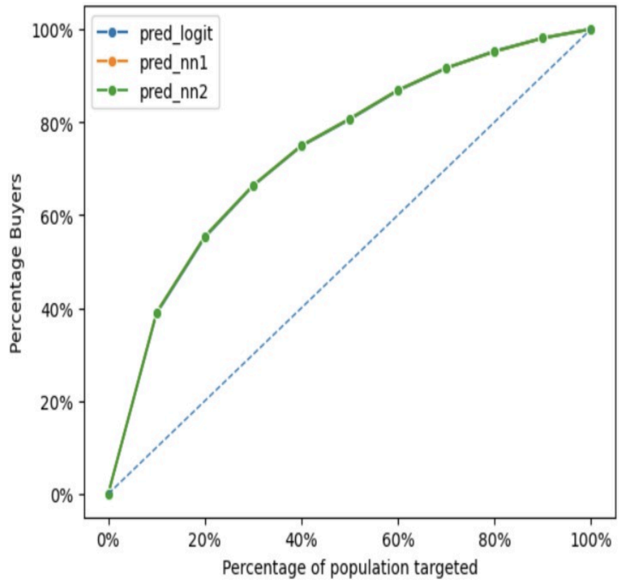
We started by selecting training data ('training = 1') from ‘intuit75k’ and included all available variables in the logistic regression model. We then systematically removed variables that were not significant, starting with ‘id’ and ‘sex’, since their p-values were high and they did not improve model performance as shown to us by the Pseudo R-squared values which remained unchanged when these variables were dropped.

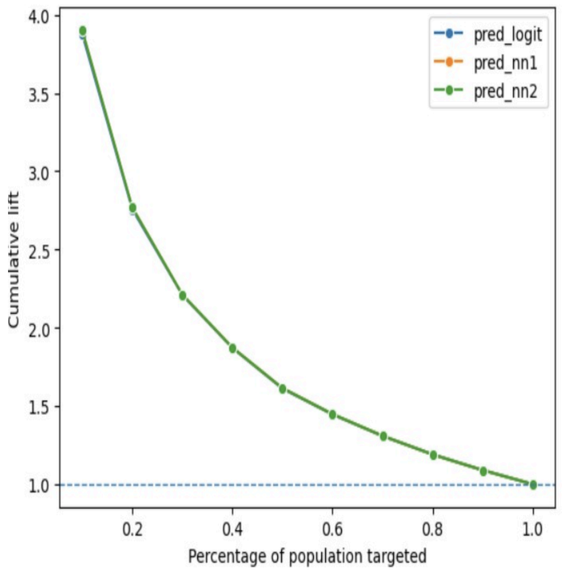
To validate feature importance, we used LASSO regression to generate coefficients for the remaining variables. From this, we were able to deduce that ‘sinepurch’ has a negligible effect on the model. We further plotted a permutation importance graph to verify this deduction. We also examined the correlation matrix and observed that the ‘sincepurch’ variable exhibited multicollinearity with ‘version1’ and ‘upgraded’. To refine the logistic model further, we then removed the ‘sincepurch’ variable to arrive at the final variables of ‘zip_bins’, ‘numords’, ‘dollars’, ‘last’, ‘upgraded’, ‘owntaxprod’, and ‘version1’. Finally, the gains graph (pred_logit) for both training and test data showed us that there was no overfit.

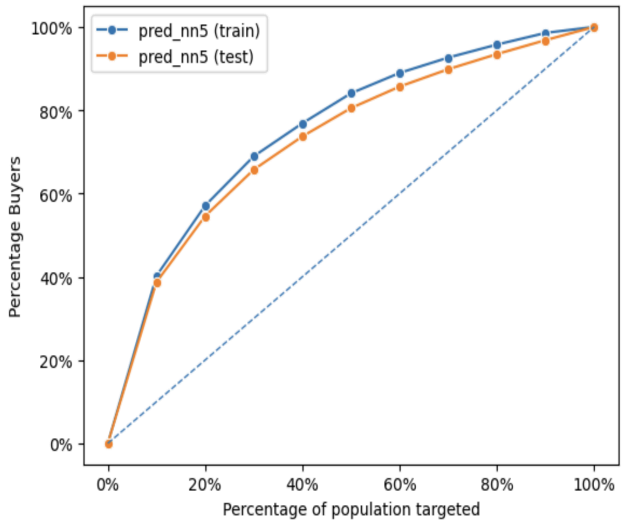


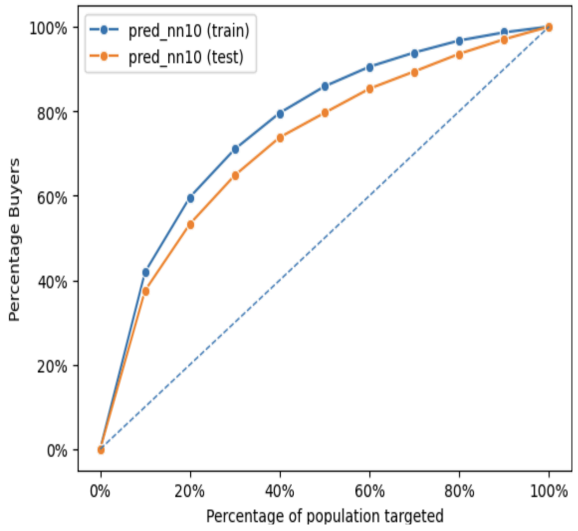
Neural Networks

After obtaining a base logistic regression model, we moved to a Neural approach where we initially used the default model with the **tanh** activation with the **lbfgs** solver to enhance predictive performance and capture non-linear relationships in the data which can then be used to improve the logistic regression. We trained the neural network using **key variables** identified from logistic regression: 'zip_bins', 'numords', 'dollars', 'last', 'upgraded', 'owntaxprod', and 'version1'.
We used multiple single hidden layer models but we noticed that the lower range of nodes was showing no improvement as compared to the logit model while the higher ones exhibited overfit. Hence we were unable to find a viable model using this solver.





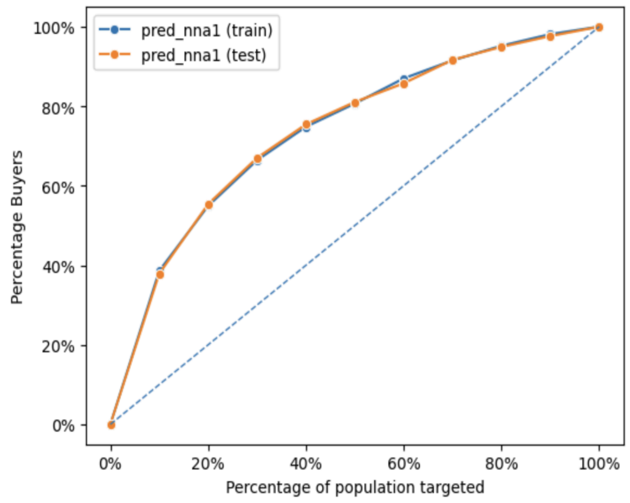


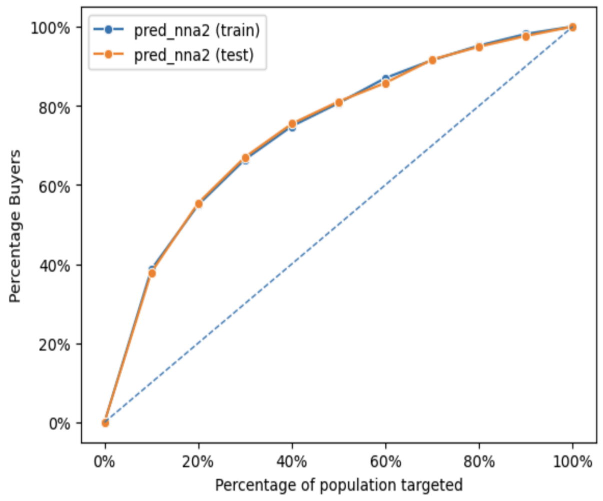


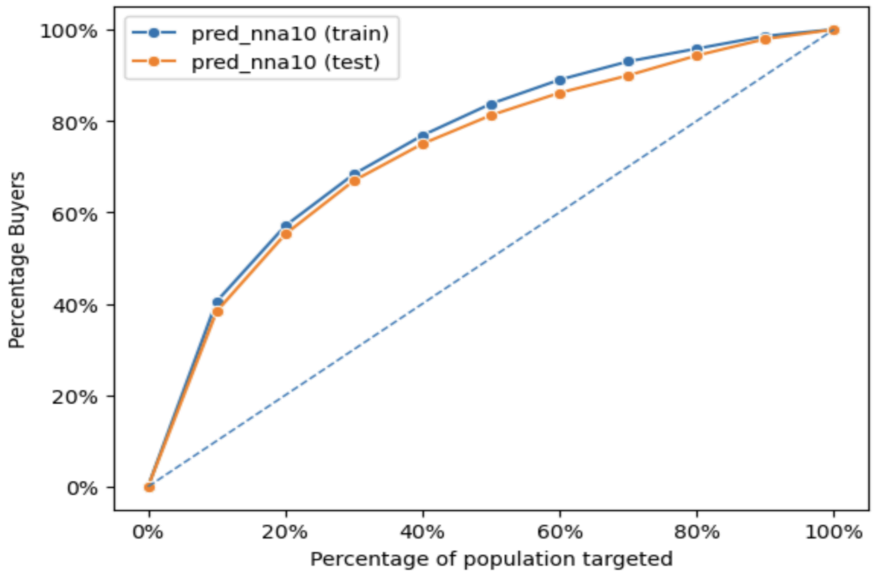
- ★ **The first two graphs showed no improvement over the logit model, while the last two graphs exhibited overfitting**

After conducting research, we decided to experiment with the **'adam'** solver combined with **'relu'** activation. Our observations revealed that models with fewer nodes performed similarly to those trained with other solvers. However, as the number of nodes increased, the model demonstrated noticeable improvements without signs of overfitting.

The **Adam solver** is an adaptive optimization algorithm that dynamically adjusts learning rates for each parameter using **momentum (past gradients)** and **RMSprop (adaptive scaling)**, improving convergence and handling sparse data efficiently. The **ReLU (Rectified Linear Unit) activation** helps neural networks learn complex patterns by introducing non-linearity while avoiding the vanishing gradient problem, making it particularly effective in deep learning models. Together, Adam and ReLU optimize training efficiency and enhance model performance, especially in high-dimensional datasets.

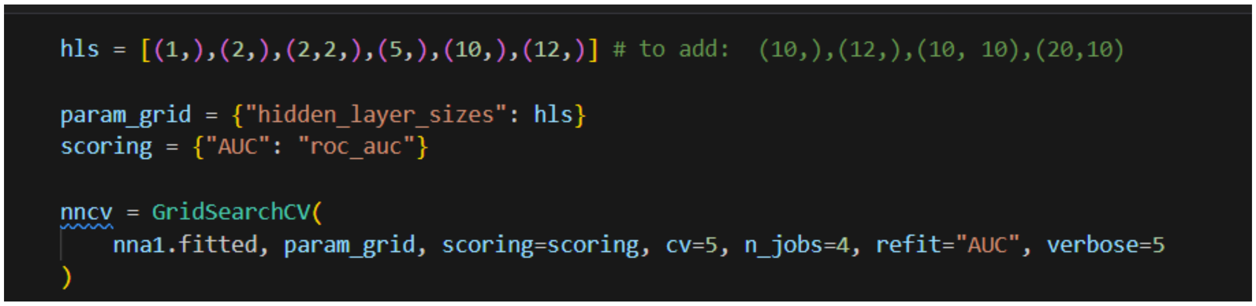






- ★ **By adjusting the solver and activation functions, the model with (10,) layers no longer overfits, making it suitable for uncovering hidden relationships**

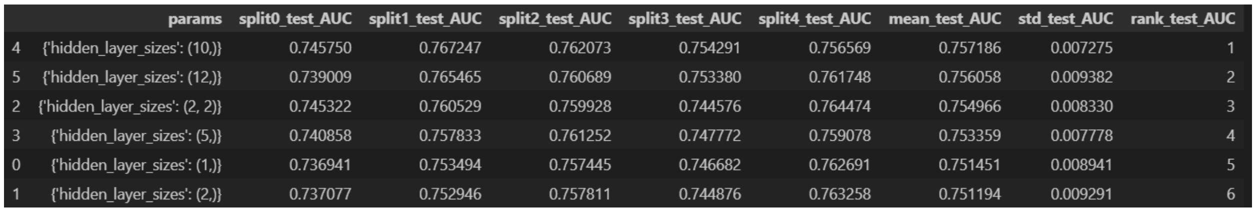
As the next step, we fine-tuned the model using cross-validation, experimenting with various hidden layer sizes to identify the optimal configuration. Through this process, we determined that the best-performing model featured a hidden layer size of **(10,)**, delivering the most effective balance of accuracy and generalization.



```
hls = [(1,),(2,),(2,2,),(5,),(10,),(12,)] # to add:  (10,),(12,),(10, 10),(20,10)

param_grid = {"hidden_layer_sizes": hls}
scoring = {"AUC": "roc_auc"}

nncv = GridSearchCV(
    nna1.fitted, param_grid, scoring=scoring, cv=5, n_jobs=4, refit="AUC", verbose=5
)
```

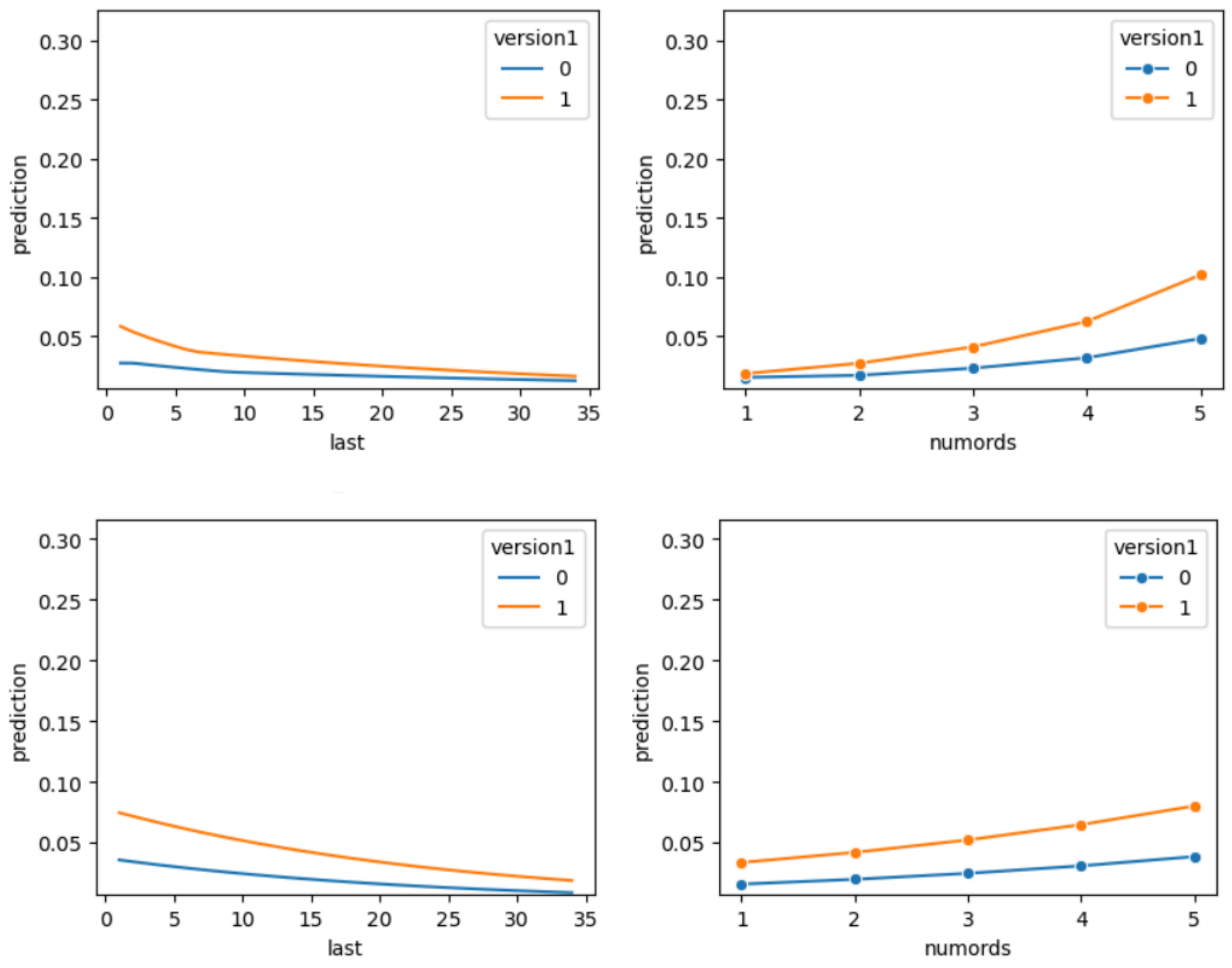


	params	split0_test_AUC	split1_test_AUC	split2_test_AUC	split3_test_AUC	split4_test_AUC	mean_test_AUC	std_test_AUC	rank_test_AUC
4	{'hidden_layer_sizes': (10,)}	0.745750	0.767247	0.762073	0.754291	0.756569	0.757186	0.007275	1
5	{'hidden_layer_sizes': (12,)}	0.739009	0.765465	0.760689	0.753380	0.761748	0.756058	0.009382	2
2	{'hidden_layer_sizes': (2, 2)}	0.745322	0.760529	0.759928	0.744576	0.764474	0.754966	0.008330	3
3	{'hidden_layer_sizes': (5,)}	0.740858	0.757833	0.761252	0.747772	0.759078	0.753359	0.007778	4
0	{'hidden_layer_sizes': (1,)}	0.736941	0.753494	0.757445	0.746682	0.762691	0.751451	0.008941	5
1	{'hidden_layer_sizes': (2,)}	0.737077	0.752946	0.757811	0.744876	0.763258	0.751194	0.009291	6

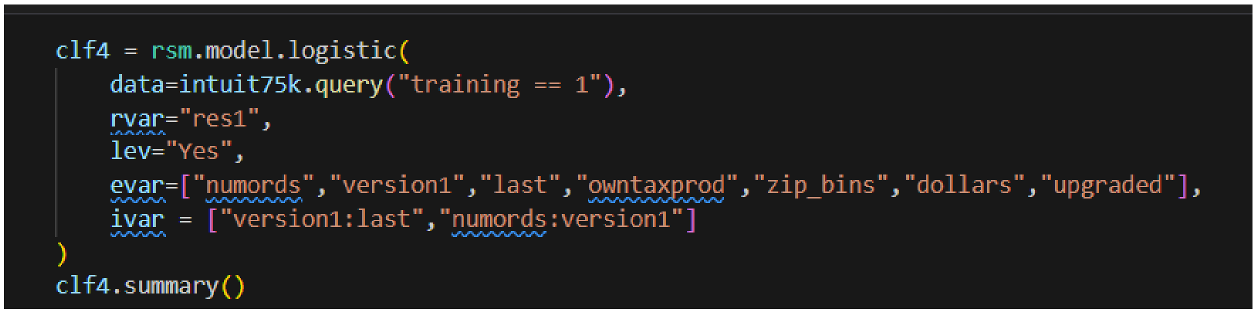
Discovering Interactions

Now that we have our base logistic regression model and final decided neural model, we proceed to discover the hidden relationships that are present in the neural model that can help improve the regression model.
As can be seen below, we notice interactions present between last & version1 and version1 and numords in the neural model which is clearly absent in the logistic model.

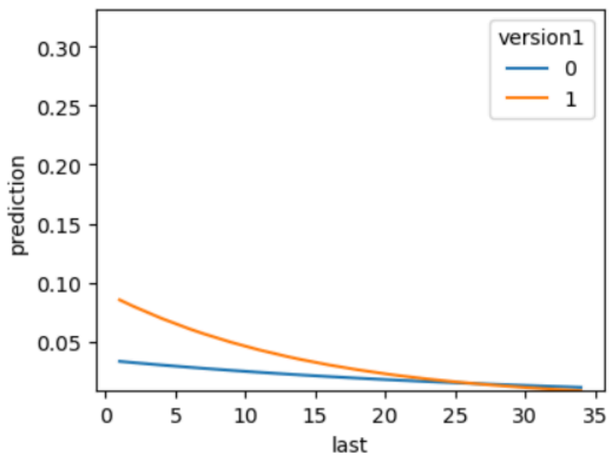
- 'version1:last' → Captures the effect of software version ('version1') and recency of purchase ('last') in influencing upgrades.
- 'numords:version1' → Reflects how the number of past orders influences response probability, considering software version history.

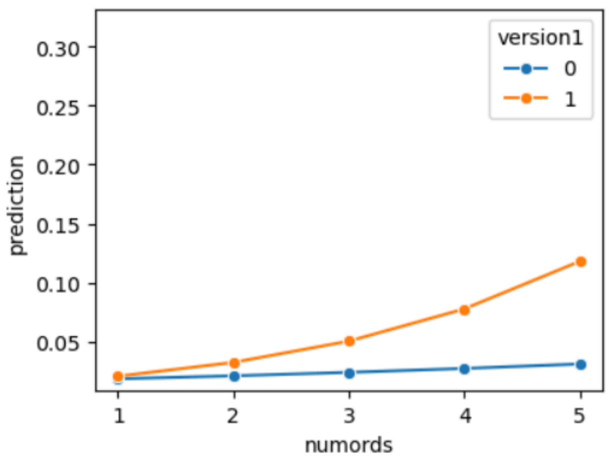


We have then further added these interactions in the logistic regression model, post which we can then see the interactions.

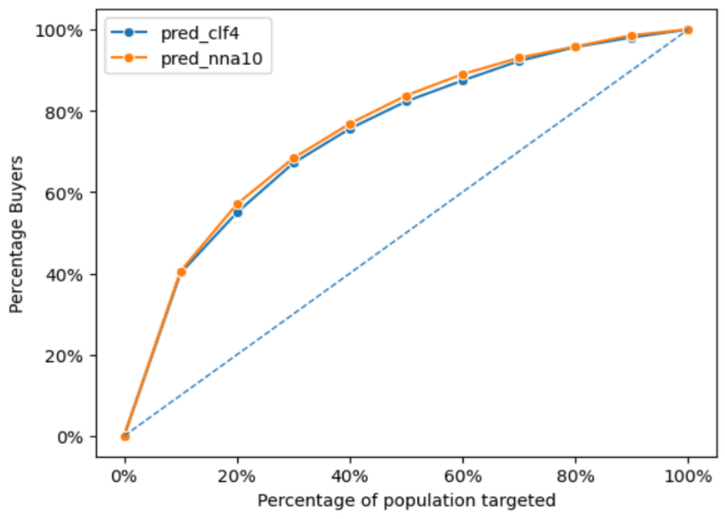


```
clf4 = rsm.model.logistic(
    data=intuit75k.query("training == 1"),
    rvar="res1",
    lev="Yes",
    evar=["numords","version1","last","owntaxprod","zip_bins","dollars","upgraded"],
    ivar = ["version1:last","numords:version1"]
)
clf4.summary()
```





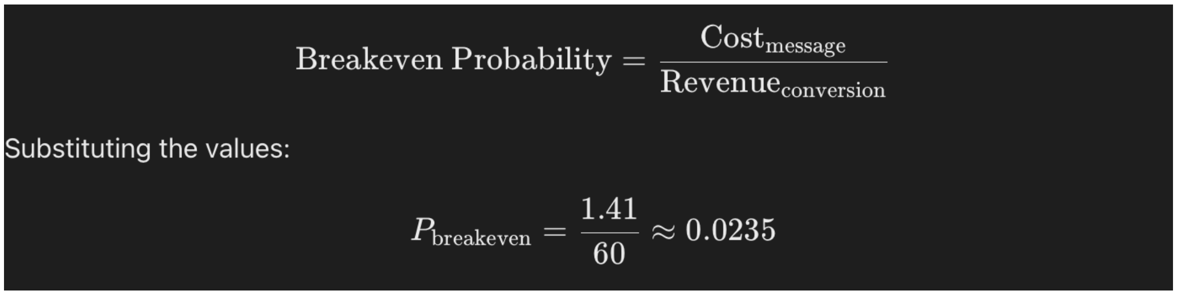
On comparing the new adjusted logistic regression model with the final neural model, we see that they are almost identical



By adding these interactions, logistic regression closely matched neural network performance, demonstrating that feature engineering can significantly improve traditional models.

Profit Analysis

For the profit analysis, we determined whether sending a marketing message to a customer would be profitable based on the breakeven probability, which was calculated as follows:

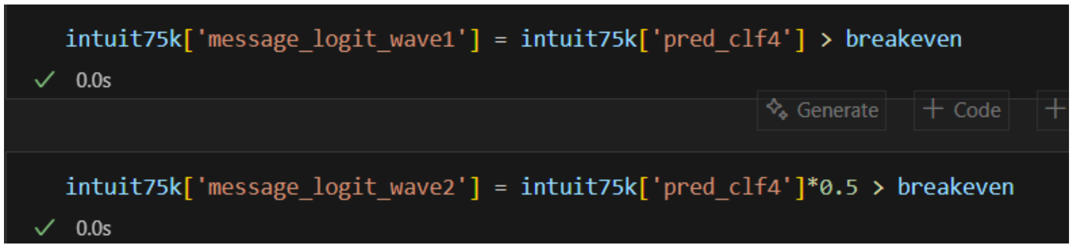


$$\text{Breakeven Probability} = \frac{\text{Cost}_{\text{message}}}{\text{Revenue}_{\text{conversion}}}$$

Substituting the values:

$$P_{\text{breakeven}} = \frac{1.41}{60} \approx 0.0235$$

Customers with predicted probability exceeding this threshold were selected for mailing.
For both waves of this campaign, the breakeven has been used as follows



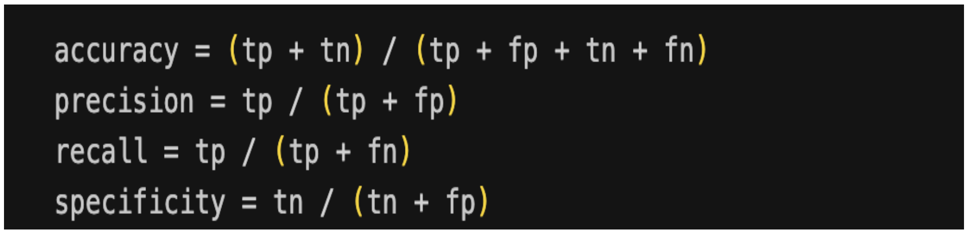
```
intuit75k['message_logit_wave1'] = intuit75k['pred_clf4'] > breakeven
✓ 0.0s
```

```
intuit75k['message_logit_wave2'] = intuit75k['pred_clf4']*0.5 > breakeven
✓ 0.0s
```

For Wave 1, the predicted probability is maintained as created, while for wave 2, we have halved this probability as it was mentioned “Usual practice would be to assume a 50 percent drop off in response from wave 1 to wave2.”

<u>For **Logistic Model:**</u>

- The **accuracy** and **precision** is calculated as follows:



```
accuracy = (tp + tn) / (tp + fp + tn + fn)
precision = tp / (tp + fp)
recall = tp / (tp + fn)
specificity = tn / (tn + fp)
```

- Accuracy: 0.4216
  Precision: 0.0687
  Recall (Sensitivity): 0.8798
  Specificity: 0.3985

→**High recall (87.98%)** → The model captures most of the actual responders, ensuring that the right businesses receive the mailing.
→**Low precision (6.87%)** → The model misclassified many non-responders as responders, leading to unnecessary mailings and increased costs.
→**Low specificity (39.85%)** → Indicates inefficiency in filtering out non-responders, contributing to campaign waste.
→**Moderate accuracy (42.16%)** → Expected due to class imbalance (most businesses do not respond)

- In **wave 1** which has been done across the entire training and test data,
  Number of Messages Sent in Wave 1: 46115
  Estimated Responses in Wave 1: 3167
  Total Profit from Wave 1: $125057.85
  Return on Marketing Expenditure (ROME) for Wave 1: 192.33%

- In **wave 2** which has been done across the test data,
  Number of Messages Sent in Wave 2: 5686
  Estimated Responses in Wave 2: 390.6
  Total Profit from Wave 2: $15419.69
  Return on Marketing Expenditure (ROME) for Wave 2: 192.33%

<u>For **Neural Model:**</u>

- The **accuracy** and **precision** is given as follows:
  Accuracy: 0.4544
  Precision: 0.0719
  Recall (Sensitivity): 0.8706
  Specificity: 0.4334

→ **High recall (87.06%)** → The model captures most actual responders, ensuring that the right businesses receive the mailing.
→ **Low precision (7.19%)** → The model misclassifies many non-responders as responders, leading to unnecessary mailings and increased costs.
→ **Moderate specificity (43.34%)** → Shows some improvement in filtering out non-responders but still results in campaign waste.
→ **Slightly improved accuracy (45.44%**) → Better than logistic regression, but still limited by class imbalance (most businesses do not respond)

- In **wave 1** which has been done across the entire training and test data,
  Number of Messages Sent in Wave 1: 43588
  Estimated Responses in Wave 1: 3135
  Total Profit from Wave 1: $126640.92
  Return on Marketing Expenditure (ROME) for Wave 1: 206.06%

- In **wave 2** which has been done across the test data,
  Number of Messages Sent in Wave 2: 5026
  Estimated Responses in Wave 2: 361.5
  Total Profit from Wave 2: $14602.58
  Return on Marketing Expenditure (ROME) for Wave 2: 206.06%

Scaled Profit Analysis

To scale the campaign, we applied the best model's performance on the full customer base, excluding the 38,487 businesses that had already responded. The results were:

- For **logistic regression (‘pred_clf4’)**:
  Number of Messages Sent in Wave 2: 202846.9936907043
  Estimated Responses in Wave 2: 13935
  Total Profit from Wave 2: $550094.52
  Return on Marketing Expenditure (ROME) for Wave 2: 192.33%

- For **neural network (‘pred_nna10’)**:
  Number of Messages Sent in Wave 2: 179301.61630135064
  Estimated Responses in Wave 2: 12895.993555674368
  Total Profit from Wave 2: $520944.33
  Return on Marketing Expenditure (ROME) for Wave 2: 206.06%

The neural network model achieved a higher **ROME (206.06%)** but with slightly lower total profit (**$5,209,44.33** compared to **$5,500,94.52** for logistic regression). However, given that the logistic regression model offers greater interpretability and a higher overall profit, the ideal choice in our case would be to proceed with logistic regression.

Final Conclusion

Both models effectively optimized marketing spend and improved conversion rates, but logistic regression emerged as the best model for final deployment. It closely matched the neural network's performance after interaction terms were added while being simpler, easier to interpret, and computationally efficient.

The two-wave strategy proved highly effective, allowing for gradual refinement of targeting efforts, ensuring that only the most promising customers were reached in Wave 2, thus reducing costs while maintaining high profitability.

Ultimately, the campaign resulted in $550,094 in total profit with a 192.33% return on marketing expenditure, proving the effectiveness of data-driven marketing optimization.