Discovering Utility From Choices

Lewis Warne: lwarne@stanford.edu

Introduction

Consumer choices are driven by their preferences. Therefore I attempt to infer a consumer's utility function by observing the choices they make.

The input to my algorithm is a set of products with know features. I use softmax regression and neural networks to output a vector of probabilities which is then used to predict the top rank item in the set of products. I compare the performance of both models across the number of training examples, and two simulated utility functions.

Data

The data in this project is simulated where the consumer has a utility U for each product i_t:

$$U(i_t) = \theta^T z_{i_t} + g(z_{i_t}) - \rho_{i_t} + \epsilon_{i_t,r}$$

I test across two utility simulations: *Linear* where $g(z_{i_t}) = 0$, and *Non-Linear* where $g(z_{i_t})$ is sum of indicator functions.

$$g(z_{i_t}) = \sum_{j=1}^{k} 50 \cdot \mathbb{1}[z_{i_t,j} > 50]$$

Notation:

- N rounds, $r \in 1, \cdots, N$, with m products in each round
- k features for each product, z_{i_t} is product i_t's features
- $Z_r \in \mathbb{R}^{m \times k}$, the features of the products in round r
- $c_r \in \mathbb{R}^m$, the one hot vector of the top ranking product
- S_r set of products available in a round r
- ρ_{i_t} price of product i_t
- $u^{(r)} \in \mathbb{R}^m$ is the predicted score or utility vector

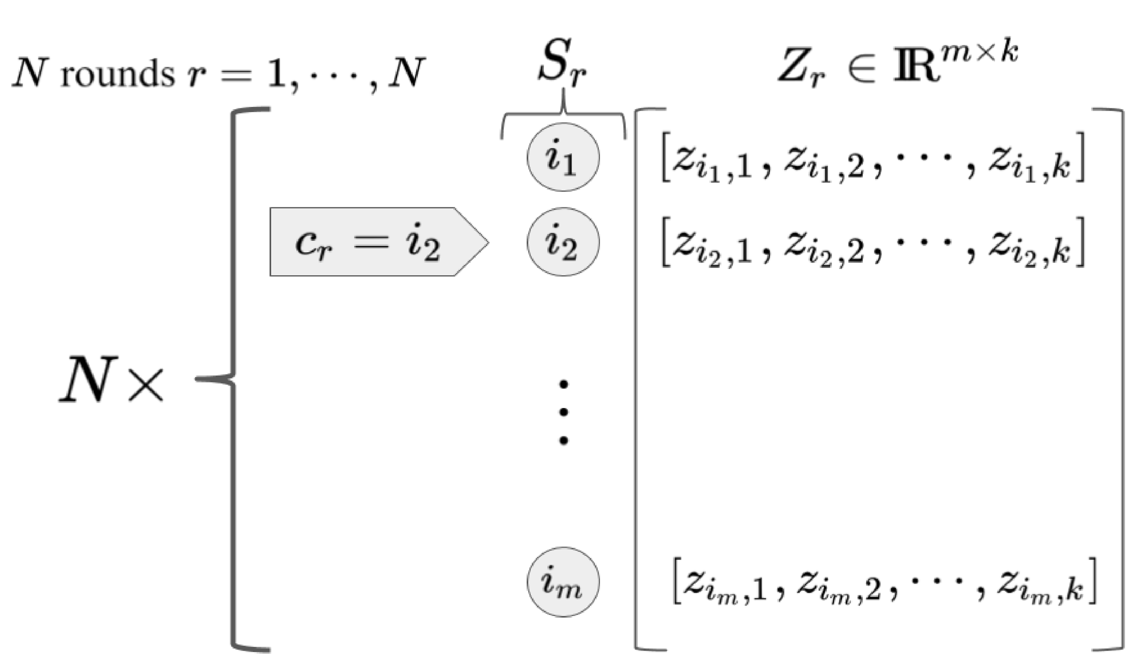
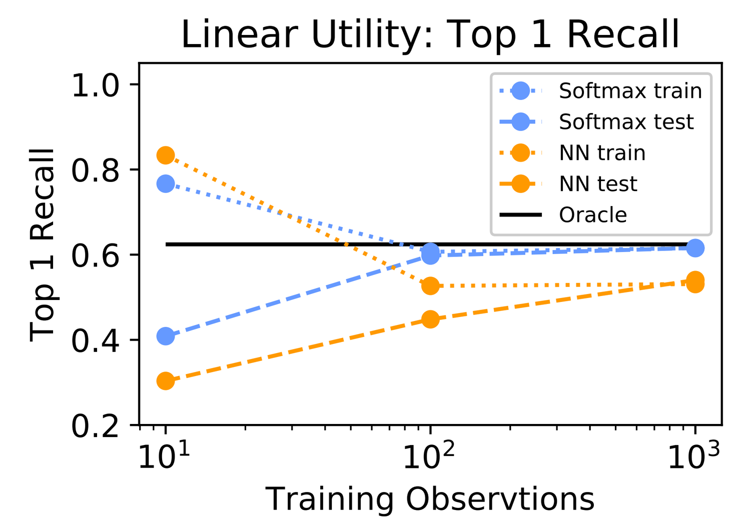
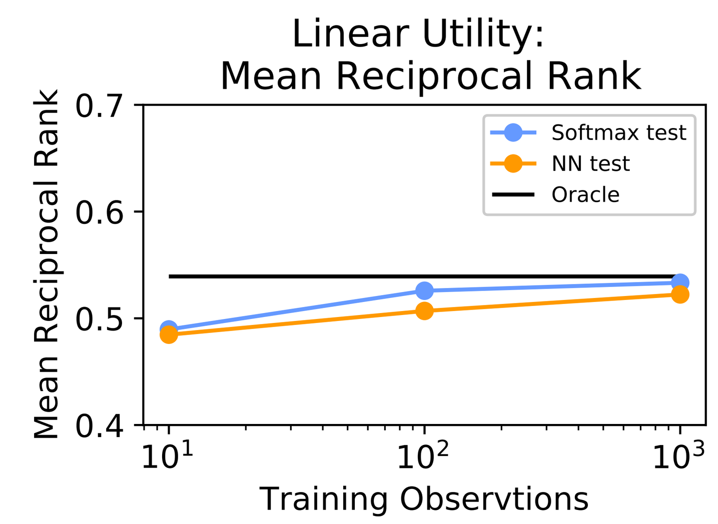


Fig. 1: Data Structure

Results

Linear Utility: Softmax regression performs well in low data regimes. If the model and features are well specified, the linear model outperforms a neural network and achives near optimal performance.

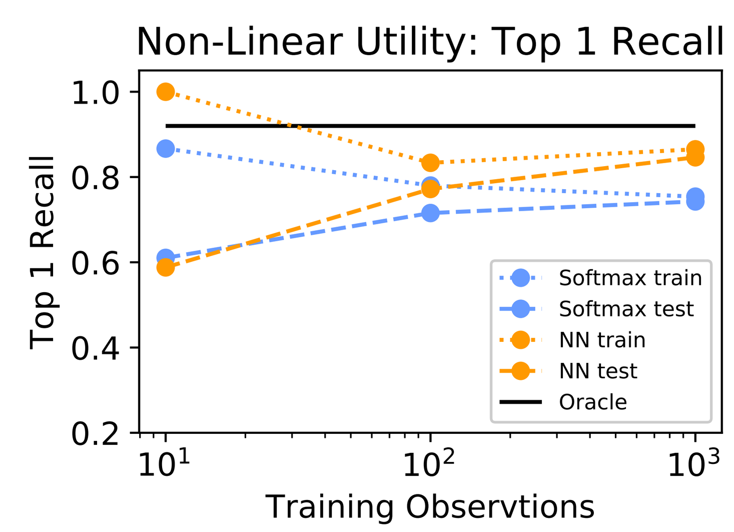


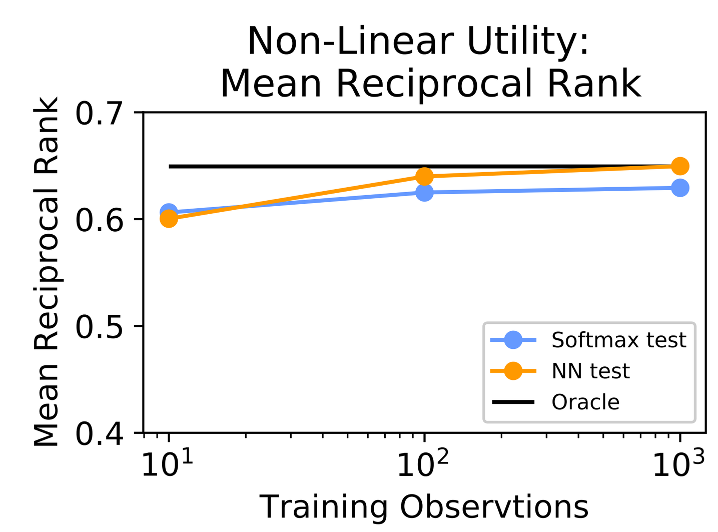


Top 1 Recall

	N	Oracle	Train	Test
	10	.6139	.7668	.4087
NN	100	.6181	.6067	.5980
	1000	.6172	.6153	.6157
	10	.6139	.8333	.3035
SM	100	.6181	.5266	.4482
	1000	.6172	.5310	.5404

Non-Linear Utility: The Neural network finds non-linear trends even in moderate size data regimes. Softmax regression fails when trying to fit a non-linear function.





Top 1 Recall

	N	Oracle	Train	Test
	10	.9178	.8667	.6103
NN	100	.9145	.7800	.7155
	1000	.9198	.7540	.7427
	10	.9178	1.00	.5879
SM	100	.9145	.8333	.7719
	1000	.9198	.8650	.8462

Top 1 Recall: What percent of top ranked predictions are correct?

$$\frac{1}{N}\sum_{r=1}^{N} \mathbb{1}\left[\arg\max_i \frac{e^{u_i^{(r)}}}{\sum_{j=1}^m e^{u_j^{(r)}}} = c_r\right]$$

Mean Reciprocal Rank: What is the average inverse predicted rank of the observed selection?

$$\frac{1}{N}\sum_{r=1}^{N} \frac{1}{\text{rank}(c_r)}$$

The test set has 10,000 observations in all cases. $m = 5$ for all rounds.

Discussion

As expected, the linear softmax regression performs well in low data regimes and when faced with a linear model. With $N = 1000$ observations it recovers the linear utility function completely and predicts as well as the Oracle model (which knows the utility scores).

The neural network suffers from over fitting in low data regimes, and does not match the softmax regression in the linear case even with more data. However, the neural network model fit the nonlinear utility function well and outperformed the softmax regression at $N = 100$ observations.

Future Work: The next steps are to apply these models to real data, which requires more complex methods like latent classes to group customers together.

Neural Network

Figure 4 shows the forward propagation of the neural network workflow. The neural net converts a products feature vector (red bar) into a utility score (red square). The NN has three hidden layers of 100 neurons with ReLU activation. Negative log likelihood loss propagates though the softmax and resizing to the neural network .

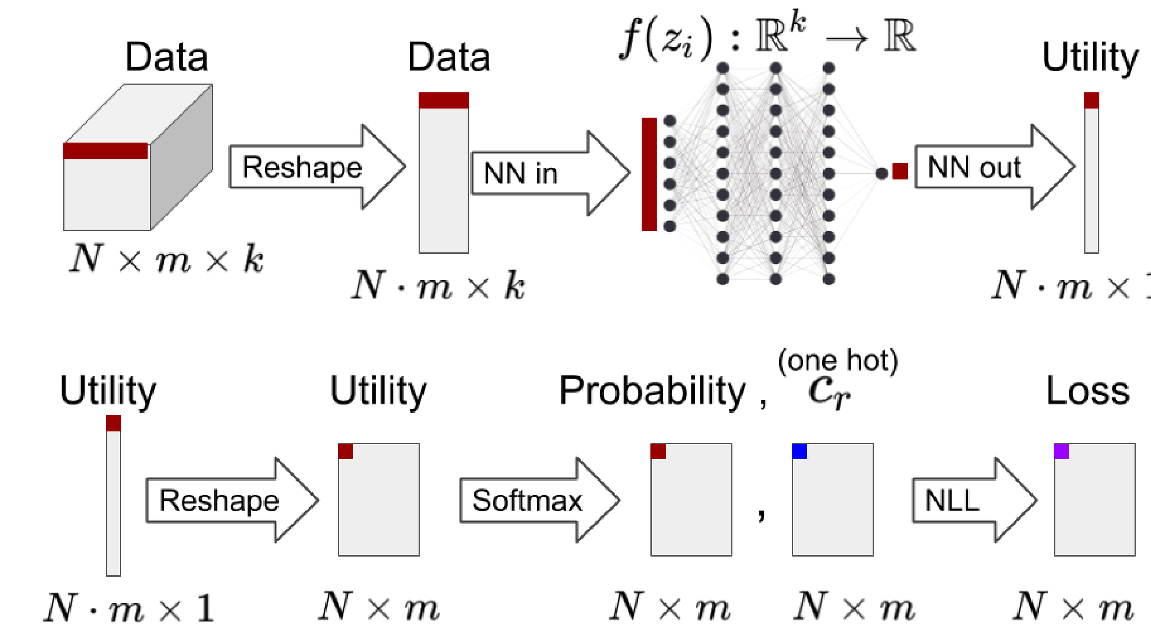


Fig. 4: NN Architecture

$$\text{NLL loss}(c_r, u^{(r)}) = -\log\left(\frac{e^{u_{c_r}^{(r)}}}{\sum_{j=1}^m e^{u_j^{(r)}}}\right)$$

Softmax Regression

I developed a gradient ascent method to maximize likelihood

$$L(\theta) = P\left[(c_1|S_1) \bigcap \cdots \bigcap (c_r|S_r) | \theta, Z\right]$$

$$\frac{\partial l(\theta)}{\partial \theta_d} = \sum_{r=1}^{N}\left(Z_{c_r d} - \sum_{p=1}^{m} \frac{e^{u_p^{(r)}}}{\sum_{j=1}^m e^{u_j^{(r)}}} Z_{pd}\right)$$

References

References

[1] Paszke, Adam, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. "Automatic differentiation in pytorch." (2017).