By the feasibility of x^*, we must have that $\sum_{(i,j)\in\hat{S}_1} c_j \le C$. Further, without loss of generality, every product has a space consumption of at most C. Therefore, $\sum_{(i,j)\in\hat{S}_2} c_j \le C$ as well, and so $\sum_{(i,j)\in\hat{S}} c_j \le C$. Next, we consider the revenue of assortment $\hat{S}$.

$$\begin{aligned}
R(\hat{S}) &= \max\left\{R(\hat{S}_1), R(\hat{S}_2)\right\} \\
&= \max\left\{\sum_{(i,j)\in\hat{S}_1} r_j P_{(i,j)}(\hat{S}_1), \sum_{(i,j)\in\hat{S}_2} r_j P_{(i,j)}(\hat{S}_2)\right\} \\
&\ge \max\left\{\sum_{(i,j)\in\hat{S}_1} r_j P_{(i,j)}(S'), \sum_{(i,j)\in\hat{S}_2} r_j P_{(i,j)}(S')\right\} \\
&\ge \frac{1}{2}\left\{\sum_{(i,j)\in S'} r_j P_{(i,j)}(S') x^*_{(i,j)}\right\} \\
&\ge \frac{\alpha}{2}\cdot\mathcal{R}'.
\end{aligned}$$

The first inequality comes from the fact that $\hat{S}_1, \hat{S}_2 \subseteq S'$, and hence the MNL purchase probabilities of each product can only decrease moving from $\hat{S}_1$ or $\hat{S}_2$ to S' . The second inequality results from the fact that

$$\sum_{(i,j)\in\hat{S}_1} r_j P_{(i,j)}(S') + \sum_{(i,j)\in\hat{S}_2} r_j P_{(i,j)}(S') \ge \sum_{(i,j)\in S'} r_j P_{(i,j)}(S') x^*_{ij}.$$

The last inequality results since $Z^*_{knap} = \sum_{(i,j)\in S'} r_j P_{(i,j)}(S') x^*_{(i,j)} \ge \alpha\cdot\mathcal{R}'$

At this point, we consider either offering the assortment S'' or $\hat{S}$. Let $\bar{S} = \arg\max_{S\in\{S'',\hat{S}\}} R(S)$. To conclude the proof of Theorem 2 we show that $R(\bar{S}) \ge \frac{1}{3}\cdot\mathrm{OPT}$. To do so, note that we have that $R(\bar{S}) = \max\{\mathcal{R}'', R(\hat{S})\} \ge \max\{(1-\alpha)\cdot\mathcal{R}', \frac{\alpha}{2}\cdot\mathcal{R}'\}$. Recall that $\alpha\cdot\mathcal{R}' + (1-\alpha)\cdot\mathcal{R}'' \ge \mathcal{R}^*$. Therefore, if $(1-\alpha)\cdot\mathcal{R}'' \ge \frac{1}{3}\cdot\mathcal{R}^* \ge \frac{1}{3}\cdot\mathrm{OPT}$, the result holds. Otherwise, $\alpha\cdot\mathcal{R}' \ge \frac{2}{3}\cdot\mathcal{R}^*$, and again, the result holds.

C. Implementation Details for Estimation Case Study

In this section, we provide the code we use for fitting the MNL and machine learning models within the case study presented in Section 3.3.

C.1. Tensorflow MNL MLE implementation

This section contains a Tensorflow python implementation of the MNL MLE problem given in (1). The input parameters have the following meaning, given that training data set contains records of τ customer arrivals.

- num_features: number of features in the data set. In our case this is 25.
- assort_size: The number of products in each offered assortment. In our case this is 6.

- offer_data: This is a assort_size $\cdot$ $\tau \times$ num_features numpy array, where each row is a feature vector X_{jt}.
- purchase_data: This is a assort_size $\cdot$ $\tau \times 1$ numpy array, where each row is a binary indicator of whether the given product was purchased by the arriving customer.
- batch_size: The number of data points to use for training in each iteration of stochastic gradient descent. We use 20,000.
- learning_rate: The learning rate of the stochastic gradient descent algorithm. We use 0.05.

```
def Estimate_MNL_Classic(num_features, assort_size, offer_data, purchase_data, batch_size, learning_rate):

    #Placeholders for offer data
    offer_mnl = tf.placeholder(tf.float32, [None, assort_size, num_features])

    #Placeholder for purchase data
    purchase_mnl = tf.placeholder(tf.float32, [None, num_features])

    #Create the variables to be estimated - the MNL feature weights
    W = tf.Variable(tf.random_normal(shape=[num_features], mean=0, stddev=0.01), name="weights")

    #Compute the log likelihood
    first_term = tf.reduce_sum(tf.multiply(purchase_mnl, W),1)
    second_term =tf.log(tf.reduce_sum( tf.exp(tf.reduce_sum(tf.multiply(offer_mnl,W),2)), 1)  + 1)
    cost = tf.reduce_sum(second_term-first_term)

    #Optimization
    optimizer = tf.train.AdamOptimizer(learning_rate).minimize(cost)
    init = tf.global_variables_initializer()

    epoch_count= 0
    with tf.Session() as sess:
        sess.run(init)
        #Stopping criterion
        while epoch_count<2000:
            num_batches = int(purchase_data.shape[0]/batch_size)
            log_like = 0
            for b in range(num_batches+1):
                if b<num_batches:
                    purchase_batch = purchase_data[b*batch_size:(b+1)*batch_size,:]
                    offer_batch = offer_data[b*batch_size:(b+1)*batch_size,:,:]
                else:
                    purchase_batch = purchase_data[b*batch_size:,:]
```

```
                offer_batch = offer_data[b*batch_size:,:,:]

            num_rows = purchase_batch.shape[0]
            _,neg_log_like = sess.run(fetches=[optimizer, cost], \
                                      feed_dict={offer_mnl:offer_batch, purchase_mnl:purchase_batch})
            log_like+=neg_log_like

        epoch_count+=1

    #Get final weights
    return sess.run(W)
```

C.2. Fitting the Machine Learning Models

This section contains a python implementation of Catboost for estimating click and purchase probabilities, where we have done no hyperparameter tuning. The input parameters have the following meaning.

- df_train: Pandas dataframe containg the training data, which has a row for each product displayed to each each customer and whose columns give the various feature values. Further, there are additional binary columns (is_click and is_buy) that indicate whether the given product was clicked and/or purchased.
- feature_list: A list of column names corresponding to the features that will be used to fit the Catboost model.

```
def Estimate_ML_Catboost(df_train, feature_list):

    #Only look at clicked products
    clicked_train = df_train.loc[df_train.is_click ==1 ,:]

    #Training data for estimating conditional purchase probs
    X_train_buy = np.array(clicked_train.loc[: , feature_list])
    y_train_buy = np.array(clicked_train.is_buy)

    #Training data for estimating click probs
    X_train_click = np.array(df_train.loc[: , feature_list])
    y_train_click  = np.array(df_train.loc[: , "is_click"])

    #Fitting the two catboost models
    model_buy = CatBoostClassifier().fit(X_train_buy, y_train_buy)
    model_click = CatBoostClassifier().fit(X_train_click, y_train_click)

    return model_buy, model_click
```

D. The Comparison of Full-feature MNL and Machine Learning Model

Table 8 All-feature Model Financial Performance

	AF-MNL	AF-ML
RevenuePerVisit (RMB)	4.79	4.64
Difference (All p-values)	0.15	
Relative Improvement	3.37%	
T-test p-value	< 0.0001	
Observations	3,152,580	3,148,217

Notes. The table reports the average financial performance, in terms of revenue per customer visit, across different algorithms during our five-day-long experimental period (September 20, 2018 - September 24, 2018).

We finished implementing and testing our MNL-based approach on all features by September 15th, 2018. Therefore, we conducted a five-day-long experiment from September 20nd, 2018 to September 24th, 2018 where customers are randomly assigned into the *the all-feature-MNL-choice-model-based approach (AF-MNL approach)* and *the all-feature-ML-based approach (AF-ML approach)* based on a unique hash number derived from the given customer's ID and an experiment ID. The AF-ML approach is exactly the same as the all-feature approach in Section 6 except that the training data is in August and September instead of February and March. Similarly, the AF-MNL approach is similar to the MNL-based approach in Section 6 except (a) the training data has advanced to August and September; and (b) the estimation process uses all features instead of the top 25 features.

Over the five days of our experiment, we observe $3,591,021$ customer arrivals from $2,247,663$ million unique customers. $1,125,381$ of these customers are randomly assigned to the MNL-choice-model-based approach on all features (i.e, AF-MNL approach) while $1,122,282$ are assigned to Alibaba's original machine learning approach on all features (i.e, AF-ML approach). The customers under AF-MNL approach collectively spend $15,114,748$ RMB during the five days, while the customers in the AF-ML approach spend $14,621,580$ RMB, an improvement of $493,168$ RMB (i.e., 3.37% during the experimental period.

Table 8 presents the GMV per customer visit generated by these two approaches on all features. The first row shows that the AF-MNL and AF-ML approaches generate RMB 4.79 and 4.64 per customer visit respectively, and the different is 0.15 RMB per customer visit; in other words, the AF-MNL approach improves the revenue per customer visit by 3.37% compared to the AF-ML approach (p-value < 0.0001). This demonstrate that the MNL-based approach out-performs the machine-learning-based approach even if both approaches are utilizing all features. We note that the machine-learning-based approach on all features is exactly the algorithm and the feature set that is used by Alibaba to recommend products prior to our collaboration. This shows that our algorithm

improves the state-of-art recommender system of Alibaba by 3.37%, which leads to the adoption of our algorithm as the main recommendation algorithm in this setting on Alibaba. We also note that this improvement based on all features (i.e., 3.37%) is more modest than the improvement based on only top 25 features (i.e., 28.0%), which may demonstrate that machine-learning-based approaches can more easily scale up to more features.