Optimizing Hotel Ranking for Higher Purchase Rates

Michael Kun Yang
Mathematics & Statistics
Stanford University
kunyang@stanford.edu

Zahid Hossain
Computer Science
Stanford University
zhossain@stanford.edu

Khaled AlTurkestani
Computer Science
Stanford University
khaled@stanford.edu

Abstract

The challenging problem of smart recommendation engines is constantly being investigated. A more specific subset of this problem that we investigate in this paper is the ranking of a small set of items based on limited information about user preferences. In this paper we present two models that we implemented for ranking result sets of hotels returned by Expedia.com in response to user-submitted search queries. The first model impelements a regularized linear regression algorithm, and the second model is a modified naive Bayes algorithm. We trained both algorithms on around 10 million datapoints then made predictions on smaller subsets of the training data.

1 Introduction

The goal in ranking is to order a set of inputs in accordance with the preference of an individual or a population. In this project we consider the ranking - or sorting - of the set of hotels returned by an online travel agent (OTA) in order to maximize purchase rates. Hotel inventory is very important since users easily jump from website to website. As such, having better rankings of hotels for specific users with the best integration of price competitiveness gives an OTA the best chance of winning the sale. The challenge of this project is to sort hotels according to the likelihood of a user to eventually book a room. More specifically, given a search query performed by a user on Expedia's hotels database and the returned result of up to forty hotels, our task is to sort the hotels according to the likelihood of the user booking a hotel.

One of the most common and effective models used in building e-commerce recommendation engines is collaborative filtering [SKKR00]. In fact, there have been several impelementations of collaborative filtering-based hotel recommendation engines [Wl10] [WS13]. Since a collaborative filtering method requires a lot of data points for each user to make good predictions by comparing users' preferences to each other, we decided to not pursue this direction in our projec. In addition, collaborative filtering has been explored many times before, and we wanted to explore other models.

2 Dataset

We were provided with a training dataset of around 10 million data points and a testing dataset of around 6.6 million data points. Each data point has 53 fields, which represent information about the search query that the user performed and one of the returned hotels. In other words, each search query is represented by up to forty data points, where each data point contains information about one hotel. Moreover, fields in the each data point have information about the search query, such as when it was performed, duration of the stay, and the number of people staying; information about the user, such as the average star-ratings they have previously assigned to hotels and the average of their pervious spending; and information about the hotel, such as its star-rating and price, and its price that is provided by up to five other Expedia competitors.

Our first task was to parse the data, discretize some variables, and transform the data points into meaningful vectors. For instance, a date field was discretized by month, where a date is represented by a twelve-field boolean vector with the value 1 assigned to the corresponding month. Similarly, a country field was discretized into a 230-field boolean vector. As a result, our original 53-field data points were transformed into 516-field x feature vectors.

3 Methods and Results

This section describes the two main methods that we investigated: 1) a regularized linear regression model and 2) a modified naive Bayes model.

3.1 Regularized Linear Regression

3.1.1 Model

Our first attempt was to regress the feature vector x against the response y, where y is the measure of the user's preference to book or click the hotel described in x. Specifically, y is defined as

$$y = \alpha \times \frac{1}{\text{position}} + \beta \times \text{booking\_bool} + \gamma \times \text{click\_bool}$$

where position = the position of the hotel w.r.t. the result list of hotels; booking_bool = 1 if the user booked the hotel; click_bool = 1 if the user clicked on the hotel; and α, β and γ are assigned weights. Note that the way we modeled y is such that it is higher if the user clicked on the hotel link, the user booked it, and/or it was listed higher in the search result. Thus, the higher the value of y, the more likely that a user will book the hotel.

We used a regularized linear regression model where our (x, y) pairs are defined as mentioned above and in the Data Section. Our squared-error loss function is defined as follows:

$$\text{Loss}(\theta) = \sum_i (y_i - \theta^T x_i)^2 + \lambda \theta^T \theta$$

where $x \in \mathcal{R}^{516}$ and $\lambda =$ penalizing coefficient.

Given the large size of our dataset (10 million data point), we implemented our learning model to run on an Apache Spark[ZCF+10] distributed system on eight cores. Running this code generates a θ. Given the θ value, we use Matlab to make a prediction.

3.1.2 Measure of error

To compute our training error, we measure the deviation of our predicted ranking for a set of n hotels that were returned for a certain search query from the actual ranking that was placed by Expedia and was included in the training set. More specifically, for the set $H = \{h_1, ..., h_n\}$ of returned hotels, their corresponding ranks $R = \{r_1, ..., r_n\}$ within the result page, and the predicted ranks $R' = \{r'_1, ..., r'_n\}$ by our model, our error ϵ is defined as

$$\epsilon = \frac{\sum_{i=1}^{n} |r_i - r'_i|}{\sum_{i=1}^{n} |i - (n+1-i)|} = \frac{\sum_{i=1}^{n} |r_i - r'_i|}{2\lfloor \frac{n+1}{2} \rfloor \Big(n - \lfloor \frac{n+1}{2} \rfloor \Big)}$$

Note that the denominator represents the deviation measure of the worst-case ranking where the hotels are completely inverted with respect to the actual ranking.

3.1.3 Results

After computing our θ vector, we used it to make a prediction over a sample of 100,000 data points from our training dataset. Using the error equation above, our computed error was 0.75. Moreover, Figure 1 shows the Q-Q plot of residues. Note that the heavy tail on the left is reasonable because of many missing values; when the hotel ranking within a result set is missing from the training data, we set the position value when computing the score y to a large number (say 1×10^7), which results in y being very small.

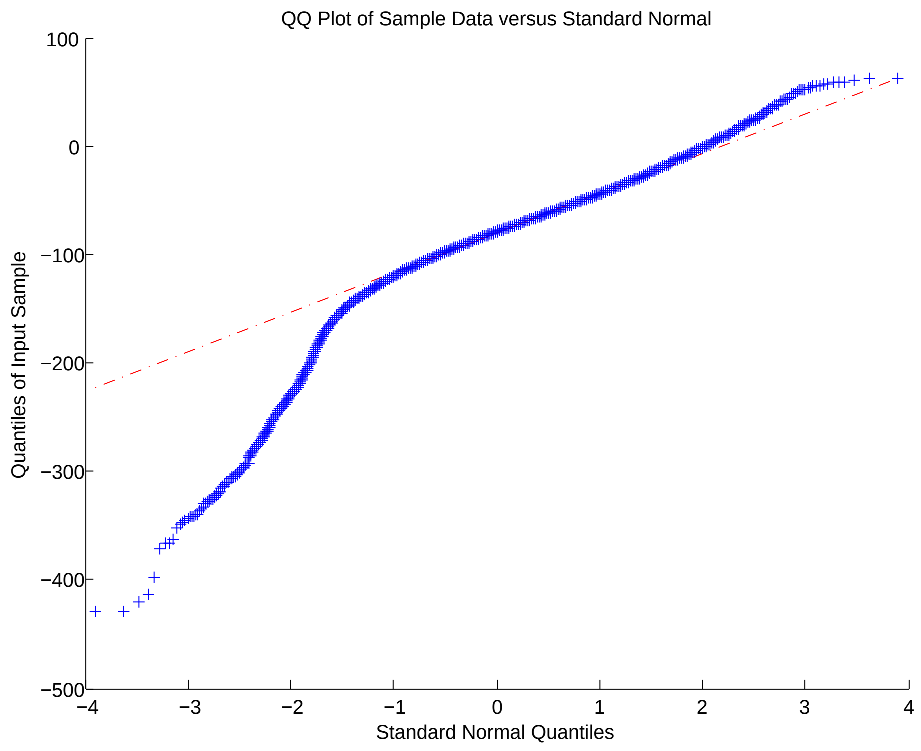

Figure 1: Q-Q plot of residues, where the heavy tail on the left is expedted due to missing hotel position values from the training set that result in very small y values.

3.2 Modified Naive Bayes

In this section we describe an alternative2 approach based on a modified version of Naive Bayes to address the hotel ranking problem. A search query yields a list of candidate hotels which we label, denoted by y, with one of the three classes, i.e. "BOOKED". "CLICKED" and "NONE" [1]. The goal is to predict the probability of a hotel's true label given a feature vector x.

3.3 Model

We make the following modifications to the traditional Naive Bayes model to lift off the strong assumption of feature independence for some selected features:

- There were features in a search query item that were clearly correlated, e.g the "price/night" (ρ_{hotel}) of the hotel in the search result and the average "price/night" ($\mu_{user,price|y}$) that a user has paid in her history. We combined these two features into one by modeling them as a Gaussian distribution on the difference, i.e.
$$\epsilon_{user,price|y} = \left(\mu_{user,price|y} - \rho_{hotel}\right) \sim \mathcal{N}\left(\mu_{user,price|y}, \sigma^2_{user,price|y}\right)$$
One would expect that this distribution, given the user has "booked" a hotel, would have $\mu_{hu} = 0$ unlike other labels. In other words; a user would most likely book a hotel that agrees with the price range that she has spent in the past.
- The rating (a number that varies from 0 to 4) of a hotel γ_{hotel} and the average rating of the hotels that a user has booked in the past $\mu_{user,rating|y}$ are also combined similarly into a Gaussian distribution of the difference, i.e.
$$\epsilon_{user,rating|y} = \left(\mu_{user,rating|y} - \gamma_{hotel}\right) \sim \mathcal{N}\left(\mu_{user,rating|y}, \sigma^2_{user,rating|y}\right)$$

[1]"BOOKED": user ends up booking the hotel, "CLICKED": user clicks on the hotel but does not book it, "NONE": user neither books not clicks on the hotel.

- Finally we suspect that some hotels might be more popular during one part of the year instead of all year round. Therefore, taking only the date into account, we could make reasonable predictions on which hotel is more likely to be picked by the user. Towards this end we modeled the date as a floating number that denotes the rank of that date within a year, e.g. 101.5 denotes 10th April at 12:00 PM of any year because its the 101th day of a year and the fractional 0.5 captures the fact that its in the middle of the day. We learned a probability distribution of this date given a hotel and a label y assuming its a Gaussian, i.e.

  $$P(date|hotel, y) = \mathcal{N}(\mu_{date|hotel,y}, \sigma^2_{date|hotel,y})$$

  Since there are finite number of hotels, in our case 136887, and three possible labels we have to learn a table of size $136887 \cdot 3 \cdot 2 \cdot$`sizeof(float)`$\approx$ 3MB for all the $\mu_{date|hotel,y}$ and $\sigma^2_{date|hotel,y}$

This way we model dependancies within some selected features. We construct our feature vector x to be a concatenation of a set of continuous and a set of discrete random variables selected from a search query item like the following:

$$x = [c_1, c_2 \dots c_a, d_1, d_2 \dots d_b], \qquad x \in \mathcal{R}^{a+b}$$

where c_i are the continuous variables and d_i are the discrete variables. All the continuous variables are modeled using Gaussian and all the discrete variables using some Multinomials, i.e. $c_i \sim \mathcal{N}\left(\mu_i, \sigma_i^2\right)$ and $d_j \sim \text{Multinomial}\left(\phi_j\right)$. Note that we do not put the feature "date" in the vector x but deal with it seperately as discussed in the following paragraph.

Given a vector x we compute the probability of the hotel taking a certain label in two separate steps and combine them assuming each step is an independent event. In the first step we compute the following probability

$$P(y|x) = \frac{P(y)\prod_i P(c_i)\prod_j P(d_j)}{\sum_y p(y)\prod_i P(c_i)\prod_j P(d_j)}$$

Next, we compute the probability of the same label but using "'date" and a given hotel by the following

$$P(y|hotel, date) = \frac{P(date|hotel, y)P(hotel|y)P(y)}{\sum_y P(date|hotel, y)P(hotel|y)P(y)}$$

We combine these probabilities assuming these two are independent events by simply taking a product:

$$P(y|x, hotel, date) = P(y|x)P(y|hotel, date)$$

Finally, given a list of hotels for a search query, we compute the above probability for a certain label and sort all hotels in the decreasing order of probabilities. Since the labels that are relevant for sorting are only "BOOKED" and "CLICKED" we only consider these two labels for the final result.

3.4 Measure of Error

To benchmark our algorithm we computed a Normalized Discounted Cummulative Gain (NDCG) [JK02] for a given search query that consists of multiple hotels as potential candidates. We used hold-out cross validation to compute NDCG and repeated the process while increasing the training set size but keeping the test set fixed to obtain a learning curve as shown in Figure 2

3.5 Results

As shown in Figure 2 the performance of Naive Bayes model plateaus out after a certain training set size which suggest that it has a bias issue. Apart from the strong assumptions of Naive Bayes model about feature independence, this is quite expected, even with our modifications, because of the following reasons:

- Our feature vector size (which is 35) is not as large as what typically has been seen to work very well with Naive Bayes model.
- For every continuous variable in the feature vector we assumed a Gaussian distribution which fails to capture any multi-modality that might be inherent in the dataset.

We also noticed that the best result was obtained when the hotels were sorted using only the probabilities of the "BOOKED" label while we did try to take some combination of "BOOKED" and "CLICKED" probabilities too.

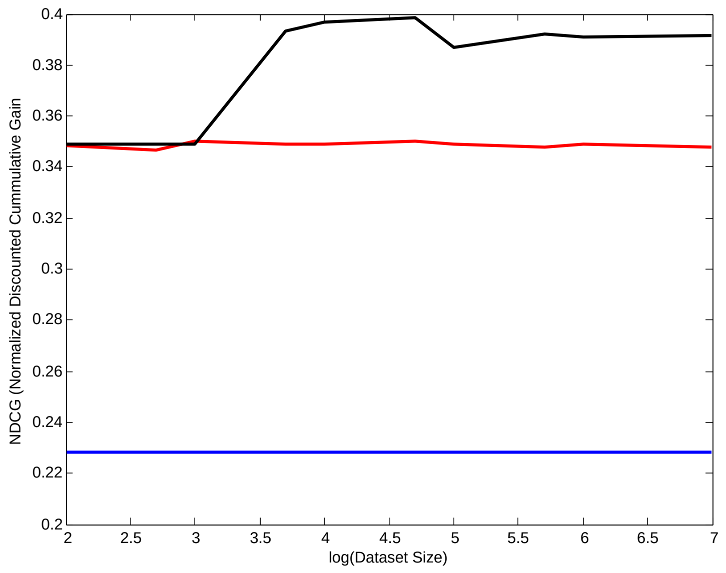

Figure 2: Learning curve of our Naive Bayes model. Each curve shows how NDCG varies with respect to the training set size (in $\log_{10}$ scale). The **black** curve is our Naive Bayes model, the **red** curve is generated when we sort hotels randomly while the **blue** curve is generated when the hotels are sorted in the worst possible manner, i.e. the most likely hotel to be booked is listed at the end.

4 Conclusion

We implemented and regularized linear regression model and a modified naive Bayes model to rank a set of hotels returned to a user for a given search query, where the objective was to sort the hotels according to the likelihood of the user booking one and, as a result, increase overall purchase rates. The linear regression model had a high error of 0.75, and the naive Bayes model performed better than random ranking. We think that further investigation is needed since our results are preliminary. For instance, in the linear regression model there are many ways of computing the score y other than the one we used that might yield better predictions. Another modification to the model would be to train it over hotels using a subset of the feature vector x, train it over users using another subset of the feature vector, and then combining the two models to predict a final score y. Moreover, the naive Bayes model could potentially be more intelligent (and less naive) by combining different features that are correlated.

References

[JK02] Kalervo Järvelin and Jaana Kekäläinen. Cumulated gain-based evaluation of ir techniques. *ACM Transactions on Information Systems (TOIS)*, 20(4):422–446, 2002.

[SKKR00] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Analysis of recommendation algorithms for e-commerce. In *Proceedings of the 2nd ACM conference on Electronic commerce*, pages 158–167. ACM, 2000.

[Wl10] GAO Hu-ming LI Wei-li. Hotel recommendation system based on collaborative filtering and rankboost algorithm. *Microcomputer Information*, 36:085, 2010.

[WS13] Qinzhu Wu and William Wei Song. A computational model for trust-based collaborative filtering. 2013.

[ZCF$^+$10] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica. Spark: cluster computing with working sets. In *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, pages 10–10, 2010.