Reinforcement Learning for Flight Ticket Pricing

Niki Agrawal and Ramnik Arora

Introduction

Airlines employ complex, secretly-kept algorithms to vary flight ticket prices over time based on several factors, including seat availability, airline capacity, the price of oil, seasonality, etc. At any point in time, a customer looking to purchase a flight ticket has the option to buy or wait (in the hope of the flight price reducing in future). However, since they lack knowledge of these algorithms, customers often default to purchasing a ticket as early as possible rather than trying to optimize their time of purchase.

However, vast quantities of data regarding flight ticket prices are available on the Internet. Through this project, we hoped to use this data to help customers make their decisions. We created an airline ticket-buying agent that tries to buy a customer's flight ticket to optimize for price of purchase.

Kayak's price predictor/Hopper/Google Flights provide some directional help in the flight booking process but we are trying to improve upon them by creating an agent that can make the purchases on our behalf.

As input, given a customer looking to purchase a particular flight ticket at X hours before departure, our algorithm outputs a decision of buy (purchase the flight ticket now) or wait (wait until the next time step to re-evaluate whether to make a purchase). The models we built include a baseline model, Q-learning, and a Deep Q-Network.

Related Work

To Buy or Not to Buy [1]: This project has the same output as our models, and it uses methods including Ripper Rule-Learning, Q-Learning, time-series analysis, and Hamlet (a combination of the previous methods). Our work employs the Q-Learning method suggested in this paper, but on a different dataset.

Global Stock Market Prediction Based on Stock Chart Images Using Deep Q-Network [2]: This paper uses a Deep Q-Network, the state of the art Reinforcement Learning method, applied to the problem of global stock market prediction (a different problem but in a similar domain).

Machine learning modeling for time series problem [3]: This project has the same output as our models. Their optimal model is AdaBoost-Decision Tree Classification, and these results are observed across 8 routes. This appears to be the state-of-the-art model for flight ticket pricing prediction.

Dataset and Features

We collected Google flights pricing data in the form of JSON files using the Google QPX Express API service. It includes six sources and destinations of flights within the US: SFO, JFK, ORD, LAX, IAH, and PHL. The data has been collected every 4 hours. For a given (flight, date-time) pair, the information we use includes:

- Carrier (Alaska Airways), Flight Number(1104), Source Airport (SJC), Destination Airport (EWR), Departure Date/Time (2019-07-01 18:45:00), Current Ticket Sale Price ($295), BookingCode_Count (6: The seats available at the current price).

The raw data has 1.57M flight prices, collected over 253 days for flights departing over a 131 day period. Given the data quality, and inconsistent collection, we had to apply a lot of data filtering. We use a subset of the collected data, which is in the date range of 3/2016-12/2017 for one-way flights on the SFO → NYC route.

Data Processing

The data was in json format (almost 300GB). For the purposes of this project we use Amazon RDS to setup a database and parse appropriate data. We constructed a label of buy versus wait for each (flight, date-time) pair by looking at the future price array until departure of each flight at each time. Given the complexity of this operation and the size of the data, we used Airflow-esque partitioning for our project. We also determined whether a flight was sold out at a given time-step by checking whether data was available for the flight at the next time-step. To do this, we had to structure our data queries so that unique flights were grouped together and data was ordered by hours before departure. Using carrier and flight number, we constructed a flight_uid to help us better represent unique flights and create equivalence classes needed for Q-Learning (detailed in Methods). We normalized features for the DQN in order to ensure they were being treated similarly in our neural network.

Below is a small sample of our data:

location_origin	location_destination	ds_departure	ds_return	id	saletotal	duration	bookingcode	bookingcodecount	cabin	carrier	flight_number	aircraft
NYC	SFO	20160802	19700101	D3yE53C0Hy1TWQZsrUOyOI00B	317.100006	365	I	5	COACH	UA	183	320
NYC	SFO	20160802	19700101	D3yE53C0Hy1TWQZsrUOyOI00E	317.100006	378	W	9	COACH	UA	637	757
NYC	SFO	20160802	19700101	D3yE53C0Hy1TWQZsrUOyOI00F	317.100006	380	W	8	COACH	UA	418	757
NYC	SFO	20160802	19700101	D3yE53C0Hy1TWQZsrUOyOI008	317.100006	363	W	9	COACH	UA	1685	757
NYC	SFO	20160802	19700101	D3yE53C0Hy1TWQZsrUOyOI009	317.100006	365	W	5	COACH	UA	912	757
NYC	SFO	20160802	19700101	D3yE53C0Hy1TWQZsrUOyOI00D	317.100006	374	W	9	COACH	UA	1978	752
NYC	SFO	20160802	19700101	D3yE53C0Hy1TWQZsrUOyOI00G	317.100006	380	I	1	COACH	VX	193	320
NYC	SFO	20160802	19700101	D3yE53C0Hy1TWQZsrUOyOI005	317.100006	348	W	9	COACH	UA	1885	777
NYC	SFO	20160802	19700101	D3yE53C0Hy1TWQZsrUOyOI007	317.100006	361	W	9	COACH	UA	1483	757
NYC	SFO	20160802	19700101	D3yE53C0Hy1TWQZsrUOyOI00H	322.100006	384	W	7	COACH	B6	415	32S
SFO	NYC	20160826	19700101	c1YjRmfwS30VkcY8fBqdhR00L	314.100006	349	W	1	COACH	B6	416	32S
SFO	NYC	20160718	19700101	0nDifGtGSCRVZXCqUsFwgV00M	314.100006	349	W	1	COACH	B6	416	32S
SFO	NYC	20160802	19700101	dAcRflulaEMR9XXTTJxcy0005	207.100006	329	S	7	COACH	VX	26	320
SFO	NYC	20160920	19700101	IJmivRkH7CJNG4HpYVAoaE00R	340.100006	335	U	5	COACH	VX	22	320
SFO	NYC	20160927	19700101	My9epUk3VFHM4HF6V8B1OK00M	207.100006	335	S	7	COACH	VX	22	320
SFO	NYC	20160728	19700101	FmMGTpYMJC0P9O95eaaanB00U	327.100006	319	I	7	COACH	VX	182	320
SFO	NYC	20160731	19700101	G9DNrlJrMh3TGq5fDarRkN00Q	412.649994	344	E	7	COACH	VX	22	320
SFO	NYC	20160916	19700101	FslbmanWAAPOUpU6wmDyt400O	207.100006	330	S	7	COACH	VX	34	320
SFO	NYC	20160526	19700101	6B3eP7F5eeFPmd5mPsolt7003	340.100006	344	U	1	COACH	VX	24	320
SFO	NYC	20160530	19700101	QMKt60bUKL6RmoZvnsXoEP00B	427.100006	339	V	1	COACH	B6	916	32S
SFO	NYC	20160523	19700101	cW2ByY22umVPY5aQ0XVYBO00N	577.099976	319	H	1	COACH	VX	182	320
SFO	NYC	20160529	19700101	TRAHnM9Toz9UYx3ev8czJJ00O	427.100006	344	V	1	COACH	B6	416	32S
SFO	NYC	20160524	19700101	JDg6sqLjYqIUXvAif85w8A00N	577.099976	335	H	1	COACH	VX	174	320
SFO	NYC	20160524	19700101	NlJYS96Wj1QUfqmVlr1EBP00N	577.099976	335	H	1	COACH	VX	174	320
SFO	NYC	20160524	19700101	7bVge8NDMVPQXHUc4gkEJ900N	577.099976	335	H	1	COACH	VX	174	320

Methods:

Baseline Model

For our baseline, the agent always selects “buy” at the earliest possible time point that the customer decides to take that flight. We decided this would be a good baseline due to the principle of risk aversion. By delaying the purchase of a flight ticket, a customer runs the risk of tickets selling out. Thus, we minimize risk aversion by choosing to buy at the earliest possible time.

Reinforcement Learning Framework

Reinforcement Learning (RL) seemed like a natural choice for our problem space. In RL models, our algorithm is an agent at a particular state. The agent has a choice of actions to make (in this case buy or wait), and after making an action it observes a reward and advances to the next state. The process repeats until the agent reaches a terminal state and can no longer continue. The algorithm’s goal is to maximize the discounted sum of rewards.

Q-Learning

The standard formula for Q-Learning is:

$$Q(a, s) = R(s, a) + \gamma max_{a'}(Q(a', s'))$$

In our Q-Learning model, we represented a state s as [flight carrier, flight number, departure_time, X hours before departure]. If the action is wait and the next state is not a terminal state, s’ = [flight carrier, flight number, departure_time, X-4 hours before departure]. We set γ = 1, so we do not discount future rewards. We keep a tabular representation of the mappings of (state, action) → Q_value.

Our reward function is as follows:

R(s, buy) = −price(s)
R(s, wait) = −Q_PENALTY if flight sells out after s
Else R(s, wait) = 0

We employ a modification of standard Q-Learning, in order to better generalize. We defined an equivalence class over states, where all states with the same carrier, flight number, and hours before departure were assigned to the same equivalence class despite having different departure dates. We redefine the Q-function as follows:

$$Q(a, s) = Avg_{s^* \sim s}(R(s^*, a) + max_{a'}(Q(a', s')))$$

We make a single iteration over the training data in each equivalence class.

Deep Q-Network
The disadvantage of Q-Learning is that it is a tabular approach, so it is difficult to generalize to a large number of features. Thus, we decided to implement a Deep Q-Network to incorporate more nuanced features, particularly that of current ticket sale price. This approach uses the standard Q-Learning formula:

$$Q(a, s) = R(s, a) + \gamma max_{a'}(Q(a', s'))$$

We define state as [current sale price, hours before departure, booking_code_count]. A state maps to a neural network, which outputs values for Q_buy and Q_wait. We chose a sigmoid activation followed by ReLU, followed by a linear output layer. Since we want to allow for negative Q_values, we decided that a linear activation function was the best choice for the output layer.

During training, the algorithm runs for n_episodes. Using our neural network, we obtain the Q-values for a given state. Epsilon is the hyperparameter that determines how often we sample an action randomly or we follow our policy, where the policy is the action associated with the maximum Q-value. We decrease epsilon over time to encourage more exploration early on in training. We then go through experience replay, in which we take minibatch_size number of (s, a, r, s') tuples and randomly sample from them to train our model.
Primary hyperparameters for our model include: number of episodes to sample on training, gamma, epsilon (which allows the model to explore early in training and sample according to Q-values later on), and minibatch_size (which determines the amount of (s, a, r, s') pairs on which to run experience replay and train a neural network). After completing this process for n_episodes, we test using our neural network to predict Q_values.

Evaluation:
To evaluate our model, we separated our data into training, validation, and test sets. Given the time series nature of this flight prices, we wanted to avoid look-ahead bias. Also, as per our Q-learning model, we wanted to learn on equivalence classes which correspond to a carrier + flight_number. Therefore, we divided our training, validation and test tests for each flight_number. We kept 65% of our departure_dates in training, 10% in dev and the remaining 25% in test set. By the very nature that we'd have more data points for flights far out, this division on departure_ds doesn't provide a 65%/10%/15% division on the number of flight points. However, this replicates the real-world scenario facing an agent who's trying to purchase these tickets.

Metrics:
Since we're trying to create a policy which can be used by an agent to purchase tickets, the profit and loss (Percent Savings) against a naive 'buy_at_first_chance' is an obvious metric. This captures the real-world considerations perfectly. Also normalizing by the initial price to convert to percent get rid of the level bias since flight price levels vary significantly.

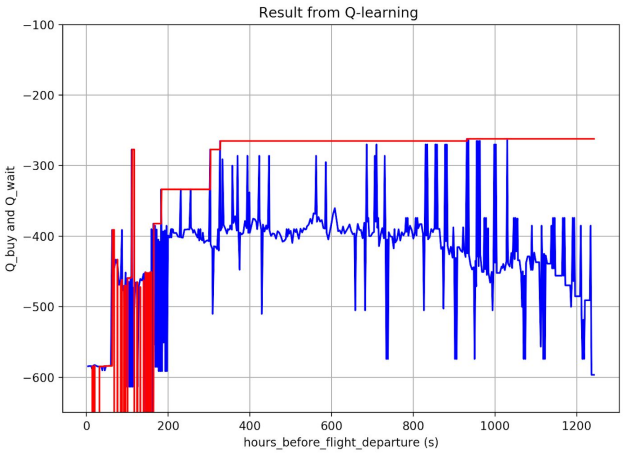
We also report the fraction of times the agent decided to “wait” (% Wait decisions), and amongst the times it decided to wait, the fraction of times it profited from its decision (% Correct wait decisions). This is primarily reported to understand how different the policies are versus a naive buy strategy and whether the wait decisions were effective.

Results

The following are the results of the Baseline, Q-Learning and the DQN model

		Percent Savings	Maximum Savings	% Wait Decisions	% Correct Wait Decisions	Total Tickets	Total Flights
Train	Baseline	0.00%	17.49%	0.00%	0.00%	58106	353
	Q-Learning	-0.79%	17.49%	89.59%	30.41%	58106	353
	DQN	7.53%	17.49%	0.00%	0.00%	58106	353
Dev	Baseline	0.00%	23.89%	0.00%	0.00%	17272	61
	Q-Learning	-0.0064%	23.89%	91.16%	31.21%	17272	61
	DQN	0.42%	23.89%	4.33%	34.28%	17272	61
Test	Baseline	0.00%	23.48%	0.00%	0.00%	21395	65
	Q-Learning	-0.62%	23.48%	91.32%	26.82%	21395	65
	DQN	-1.60%	23.48%	3.59%	31.00%	21395	65

As stated above, we have a baseline model which corresponds to an agent buying the airline ticket at first possible instant. In the Q-Learning model, our agent decides whether to {buy, wait} based on the carrier, flight number, and hours before departure. A sample of Q_buy (green) and Q_wait (red) have been shown below.



An important parameter in the model is the penalty if the flight runs out (the large spikes in red in the chart above). We observe that Q-learning predicts that several hours before departure (1200) choosing to wait is more optimal than choosing to buy, but very close to departure, buying is definitely more optimal.

The values for Q_learning were the same regardless of whether we set Q_PENALTY to be -300,000, -100,000, or -400,000. We set Q_PENALTY to be this large in order to greatly punish the sellout of flight tickets. As a result, the agent did not encounter any sellouts during testing.

Since the Q-Learning model is unaware of the price at the point of decision making, we obtained errors whereby the model would wait for a better fare even when mathematically the fare could not have become lower (see [5]

for an explanation of airline pricing). Therefore the state needed to be more nuanced, but including prices in the state would have made the Q algorithm learn from sparse data. Therefore, we implemented DQN with additional features like current_price.

As for our loss function in order to train the neural network, we chose *mean absolute error* versus the more common *mean squared error*. This decision was made to account for the fact that an error of $2 is not four times worse than an error of $1 for a ticket-buying agent. This change definitely improved the results for the DQN on the dev set.

For DQN, we changed Q_PENALTY to be 300. This is much smaller than what we used for Q-Learning, because we are only iterating over one (s, a, r, s') tuple at a time rather than putting them all in an equivalence class. The DQN did not encounter any sellout states during testing, suggesting that our Q_PENALTY was effective at avoiding sellouts.

Small Dataset Results:

		Percent Savings	Maximum Savings	% Wait Decisions	% Correct Wait Decisions	Total Tickets	Total Flights
Train	Baseline	0.00%	16.23%	0.00%	0.00%	7402	51
	Q-Learning	-15.44%	16.23%	86.14%	36.34%	7402	51
	DQN	0.00%	16.23%	0.00%	0.00%	7402	51
Dev	Baseline	0.00%	26.25%	0.00%	0.00%	2720	10
	Q-Learning	-2.24%	26.25%	93.46%	33.40%	2720	10
	DQN	0.00%	26.25%	0.00%	0.00%	2720	10
Test	Baseline	0.00%	23.77%	0.00%	0.00%	3183	10
	Q-Learning	10.17%	23.77%	94.85%	31.00%	3183	10
	DQN	0.00%	23.77%	0.00%	0.00%	3183	10

Above are our results for a single flight_uid (carrier + flight number) with a variety of departure date_times. In these results, Q-Learning has a significant improvement in performance over baseline (10% more savings) and DQN has the same performance as baseline (meaning that it always converges to a decision of "buy"). However, the results on this limited dataset display large variance, with the Percent Savings for Q-Learning varying greatly across our training, dev, and test set. Thus, these results suggest that our models may have improved performance if they are more nuanced, with access to a larger set of features.

Conclusion and Future Work

In this work we were not able to replicate the strong results from [1] despite replicating their Q-learning implementation. This is despite the fact that we had more data and a larger number of features. One hypothesis is that the airline pricing industry has become more sophisticated since the reference paper was published, so Q-learning can no longer capture the complexity of it. However, the opportunity size is still remains with wild fluctuations in airline prices.

It is important to note that in our setup, we haven't captured the nuances of real-world perfectly. In particular, when an individual searches for flights from SFO -> NYC, they are (mostly) indifferent between a number of individual flights meeting their search criteria. Therefore, the agent would ideally make the decision of "buy" or "wait" on a complete route and not an individual flight, as we've modelled. This will be an interesting and important area of exploration for future work.

The Github repo is listed in [6].

References

1. To Buy or Not to Buy:Mining Airfare Data to Minimize Ticket Purchase Price
2. Global Stock Market Prediction Based on StockChart Images Using Deep Q-Network
3. Machine learning modeling for time series problem:Predicting flight ticket prices
4. A regression model for predicting optimal purchase timing for airline tickets
5. Computational Complexity of Air Travel Planning
6. Github Repo: https://github.com/launcelotgobbo/cs229

Contributions

Most of the algorithm development / research / testing was done via pair programming or coordinating extensively. In particular, Niki contributed more heavily on the poster presentation, final paper while Ramnik handled the data engineering and manipulation.