Predicting low airfares with time series features and a decision tree algorithm

Author: Jonatan Krook
Supervisor: Patrik Andersson

Department of Statistics, Uppsala University
Spring 2018
Abstract

Airlines try to maximize revenue by letting prices of tickets vary over time. This fluctuation contains patterns that can be exploited to predict price lows. In this study, we create an algorithm that daily decides whether to buy a certain ticket or wait for the price to go down. For creation and evaluation, we have used data from searches made online for flights on the route Stockholm – New York during 2017 and 2018. The algorithm is based on time series features selected by a decision tree and clearly outperforms the selected benchmarks.

Keywords: Machine learning, flight tickets, price prediction.
Acknowledgements

This thesis was made possible through the kind cooperation of Mattias Nyman, CEO of Flygresor.se. I would like to thank him for giving me the opportunity to work with these exciting data.
## Contents

1 Introduction ......................................................... 1  
   1.1 Previous Work ................................................. 1  
   1.2 Purpose ........................................................ 2  
   1.3 Expressions ................................................... 2  
   1.4 Disposition ................................................... 2  

2 Theory ............................................................. 3  
   2.1 Price Fluctuation .............................................. 3  
   2.2 Data .......................................................... 4  
   2.3 Attributes .................................................... 6  

3 Method ............................................................ 7  
   3.1 Regression Tree ............................................... 7  
   3.2 Decision Tree ................................................ 8  
   3.3 Benchmarks ................................................... 10  

4 Results ........................................................... 12  
   4.1 Final Tree and Algorithm .................................. 12  
   4.2 Performance .................................................. 14  
   4.3 Examples ..................................................... 14  
   4.4 Conclusion ................................................... 17  
   4.5 Future Work .................................................. 17  

5 References ....................................................... 19
1 Introduction

Airlines have always been in the forefront of the use of advanced strategies to maximize revenue. Ticket prices vary over time according to models based on a wide range of variables, many of which are unavailable to the consumer (CNN 2017). Although simple rules of thumb can help keeping a budget, they do not in a satisfying way address the complex nature of airfares.

In 2016 alone, a total of 3 796 million passengers chose flying as a means of transportation worldwide (Trafikanalys 2018, 2), and the number of passengers has been increasing almost every year since 1970 (World Bank 2018). As the popularity of flying grows, so does the need for ways to find cheap tickets, and thus the business possibilities for anyone with relevant forecasting tools at hand. Some flight price comparison websites already provide such services, but how rare this is reveals what a difficult task it is.

In this study, we will examine price fluctuations for round trips on the route Stockholm – New York during 2017 and 2018 and use the obtained knowledge to create an algorithm that daily decides whether to buy a certain ticket or not. We approach this task in a previously untested way.

1.1 Previous Work

Among the studies written on airfare prediction, that of Etzioni et al. (2003, 126) is one of the most influential. They compared the performance of different methods and based their predictions on the observed price, the number of days until departure and some nominal variables fixed over time, like the airline. Among other things, they concluded that for this purpose, rule-based methods outperform traditional ARMA-models.

Groves and Gini (2013, 154) compared several approaches where predictions were made by tracking the day of the week and the number of days until departure. They also tracked the prices offered by different airlines trafficking the route, all of which were used to find exploitable patterns. They compared rule-based methods to regression-based ditto, which in their case performed equally well.

Santana, Mastelini and Barbon (2017, 25) made use of a regression-based approach where they like many before them compared prices of different airlines, but also made use of e.g. the lowest price in the near past.

Previous work has shown that airfares fluctuate by discoverable patterns. However, most previous analyses base their predictions on either the price at the time of purchase or a set of fixed nominal variables, or both. To our knowledge, no attempts have been made to create a rule-based algorithm
that mainly relies on the price fluctuation for its recommendations, a gap
we intend to fill.

1.2 Purpose

The purpose of this study is to investigate if an algorithm, based primarily
on time series features selected by a decision tree, can make cost-effective
recommendations about when to purchase a flight ticket. The study focuses
on the selection of time of purchase in relation to a day of departure selected
in advance.

1.3 Expressions

An attribute is what many recognize as a variable, such as "the number of
days until departure is five" (Fürnkranz, Gamberger and Lavrač 2009, 28).
As in the literature, we use the term attribute when discussing properties
of decision trees, and variable otherwise.

A feature is a statement about an attribute, such as "it is true that the
number of days until departure is less than seven" (Fürnkranz, Gamberger
and Lavrač 2009, 66).

The term rule is used in many different ways, but we will explicitly use
it to refer to one or more features used to make decisions.

Algorithm and decision tree are often interchangeable. We will adhere
to what we have seen in the literature and use the former in more general
terms and the latter when covering technical details.

1.4 Disposition

In Section 2, we describe on what assumptions we base our method. We also
describe data, and how they are dealt with and why. In Section 3 we describe
how the decision tree is constructed and how we evaluate the performance
of the algorithm. In Section 4 we present the results and discuss the future
work that can be done based on this study.
2 Theory

2.1 Price Fluctuation

Visual inspection reveals that the price for a certain day of departure may vary from day to day, as does the direction of the change. In some cases, no price trend seems to occur. Mantin and Koo (2009, 1022) observed that airfares generally decrease until 2 – 3 weeks prior to departure, after which they increase steadily. A bowl-shaped trend is sometimes visible, but not necessarily with a low within that time span. Other times there is a clear upward trend from the beginning.

A plot of the average prices for given number of days until departure, Figure 1, reveals that no day stands out as clearly cheaper than the others and that the price tends to increase with time.

We assume that these properties are not random occurrences in the training data, but relevant ditto that can be used to predict ticket prices in general.

Figure 1: Mean prices by number of days until departure for route Stockholm – New York. Prices used to calculate means have been standardized such that the cheapest ticket for a certain day of departure is priced 1.0.
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>trips.iata</td>
<td>Route notation by IATA standard</td>
</tr>
<tr>
<td>from</td>
<td>Outbound search</td>
</tr>
<tr>
<td>to</td>
<td>Inbound search</td>
</tr>
<tr>
<td>searchdate</td>
<td>Day and time of search</td>
</tr>
<tr>
<td>leavedate</td>
<td>Day of departure</td>
</tr>
<tr>
<td>homedate</td>
<td>Day of return</td>
</tr>
<tr>
<td>adults</td>
<td>Number of adults the search was for</td>
</tr>
<tr>
<td>children.ages</td>
<td>Ages of children the search was for</td>
</tr>
<tr>
<td>price</td>
<td>Lowest available total price</td>
</tr>
</tbody>
</table>

Table 1: Variables in raw data with descriptions.

2.2 Data

The raw data consists of all searches made for the route Stockholm – New York at the flight price comparison website flygresor.se between January 1st, 2017 and April 15th, 2018, each with the associated variables presented in Table 1.

From this set, we keep only the observations that meet the following criteria:

1. The day of departure is April 16th, 2017 or later.
2. Day of search and departure are no more than 105 days apart.
3. Trip is a round trip with in- and outbound flights between the same airports.
4. None of the tickets searched for were for children.
5. Route searched for was STO – NYC or ARN – NYC.

Through criteria 1 and 2, we manage to cover departure days for a full year while simultaneously maximizing the time during which we cover the price fluctuation. Flights that do not fulfil criterion 3 could be subject to extra fees that make comparison between flights harder. It is difficult to calculate a fair price per person when at least one ticket is for a child. Because of this and the fact that these searches make up a very small fraction of the data, we exclude them.

After calculating the price per adult for each search, the median price for every unique and available combination of departure date and search day
is extracted. These prices are used to create the 365 time series on which we conduct our experiment.

Including searches made for both STO – NYC and ARN – NYC and treating them equally could be problematic since they could result in different lowest prices. A departure from STO could be any airport in the Stockholm area, whereas ARN is Arlanda airport specifically. On average in our data, tickets searched for from ARN are 222 SEK more expensive than from STO. There is thus a risk that the algorithm learns to identify price differences caused by the search terms, and not those caused by the airlines’ pricing models.

Although possibly problematic, including the smaller of the search terms, ARN – NYC, with about 274 000 searches, brings the total number of searches up to about 620 000. This in turn allows for time series with considerably fewer missing values and in turn a better analysis.

The extent of the data does not allow taking both the day of departure and return into account, as it would yield time series too damaged by missing values. This could be problematic too, but we have not found that the number of days between departure and return has any practical impact on the price.

It is intuitive and preferable to let the mean price represent the "real" ditto for a given day. However, data do not contain information about if the search was made with some constraint as to during what time of the day the flights may depart. If so, the lowest price for the search in question could be increased since cheaper tickets are excluded. The median is thus used since it is more robust to extreme deviations.

Randomly sampled, we divide the time series into test and training samples sized one and two thirds respectively. The more price behaviour changes over time, the worse the algorithm performs in practice since past flights reveal less about future ones. Since the subsets are sampled randomly, we do not use the past to predict the future but a mixture of the two in both cases. This means that to the same extent that the future is different from the past, algorithm performance is overrated. Given our data restrictions, letting training data lay before test data in time would take this fact into account, but could at the same time bypass seasonal differences. Our assessment is that if any of the differences mentioned above exist, the seasonal ones are larger, and thus that sampling at random yields the fairest results.
2.3 Attributes

The attributes we extract and feed the decision tree are listed in Table 2 and aim at capturing the different properties of price fluctuation described in Section 2.1. We borrow Attribute 1 from a study by Wiens, Guttag and Horvitz (2012, 4) on classification of time series, a task similar to our. Attribute 2 does not have a specific origin, other than that it is intuitive to include. Attributes 1 – 3, the latter a combination of the former ones, aim at capturing short term variation. Attribute 3 is negative or positive depending on the direction of the change, and 0 if the price is the same as the previous day. Long term variation is captured by 4 and 5, attributes with infinite memory. Attribute 5 is inspired by the so-called Death Cross from the area of finance, where the use of moving averages are said to reveal turns in trends (Nasdaq 2011). Attributes 2 – 5 are calculated such that they take the value 0 if there is no difference from the reference point. We include long and short term standard deviations, 6 and 7, not based on what we learned in Section 2.1, but since they were useful to Wiens, Guttag and Horvitz (2012, 4), and since it makes sense to assume that the intensity of the variation bears useful information. The standard deviations are calculated on time series standardized such that the first observation has the price 1.0.

We also track the number of days until departure. Although independent of price fluctuation, we include it since it so heavily affects the price.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>run</td>
<td>Number of days in a row price has either increased or decreased</td>
</tr>
<tr>
<td>2</td>
<td>diff</td>
<td>Price difference relative to previous day</td>
</tr>
<tr>
<td>3</td>
<td>diff.run</td>
<td>Price difference relative to beginning of current run</td>
</tr>
<tr>
<td>4</td>
<td>diff.start</td>
<td>Price difference relative to first observed price</td>
</tr>
<tr>
<td>5</td>
<td>diff.ma</td>
<td>Price difference relative to average price</td>
</tr>
<tr>
<td>6</td>
<td>sigma</td>
<td>Standard deviation of price</td>
</tr>
<tr>
<td>7</td>
<td>sigma.7</td>
<td>Standard deviation of price last seven days</td>
</tr>
<tr>
<td>8</td>
<td>days.depart</td>
<td>Number of days until departure</td>
</tr>
</tbody>
</table>

Table 2: Attributes extracted from points in time series. No attributes make use of "future" prices when calculated. Missing values in time series do not reset attributes.
3 Method

We construct the decision tree by following a method inspired by that of creating a regression tree. In this section, we will first describe how a regression tree is constructed, and then how this procedure is changed to suit our purposes.

3.1 Regression Tree

The objective of a regression tree is to repeatedly group the observations and assign all observations in the same group some response value. Observations are grouped by whether they match some feature (or set thereof) or not. As the tree is gradually constructed, the selected feature to divide by is that which given the current structure of the tree minimizes the residual sum of squares across all groups. The response value is the mean within each group. The grouping stops when some stopping rule is triggered. As seen in Figure 2, this procedure can be illustrated by a tree-like flow chart, hence the name. The initial feature constitutes the root of the tree, the intersections are junctions and the end points are leaves. (Hastie, Tibshirani and Friedman 2009, 305).

![Figure 2: Example of a regression tree. The dependent variable is satisfaction with a flight on a scale from 1 (low) to 10 (high), given the number of babies on board and food quality.](image-url)
3.2 Decision Tree

Since we want the tree to tell us whether to buy a ticket or not, we want the response values to be *buy* or *wait*. The response is *buy* if the feature preceding the leaf is true and *wait* otherwise. Furthermore, we are not interested in homogeneous clusters of observations, but cheap flights. We thus let the feature that on average yields the lowest purchase price be the criterion when expanding the tree. In this context, a feature is a combination of some attribute $X_i$, some threshold $c_i$ and some sign $\sim$ that is either $<$ or $>$. We enable comparison between prices that fluctuate at different levels by measuring the price as a fraction of the lowest observed price for each day of departure. That is, we let the feature that across all $n = 1, 2, \ldots, N$ days of departure minimizes the loss function

$$\sum_{n=1}^{N} \frac{price_i}{\min(price)_n}$$

be the feature by which the tree is expanded, where the purchase is made on the $i$:th day. By applying these changes, we do not get a tree that groups observations, but one that evaluates whether an observation should result in the action *buy* or *wait*. In our case, the observations are the points in the time series which all have the associated attributes presented in Table 2.

Since we have time series and not cross-sectional data, unlike when building regression trees, we cannot minimize the loss function by ordinary optimization techniques. Instead we evaluate a range of features at each junction and choose that which yields the lowest average purchase price. We evaluate all combinations of attributes, signs and some set of thresholds adapted to the attribute. It is crucial that for each evaluated feature, we get a purchase on all days of departure. If not, the loss function takes a missing value, which in turn makes comparison between features impossible. We thus from the beginning institute a rule that always decides to purchase seven days prior to departure, the same *last resort rule* used by Etzioni et al. (2003, 125).

When going through this evaluation procedure for the first time, we get the root feature found in Equation 2.

```latex
\text{if diff.start} < -0.064 \text{ then } \text{BUY \ else WAIT} \hspace{1cm} (2)
```

In other words, if we were only allowed to use one feature to decide whether to *buy* or *wait*, we would choose the feature in Equation 2 since it yields the lowest average price when used on the training data. The
algorithm would then extract the `diff.start` attribute from points in the time series, compare its size to the threshold, and decide `buy` if it is smaller than \(-0.064\), i.e. when the price has gone down by more than 6.4% compared to the initial observation.

We now see if the performance of the tree can be improved by adding a feature to the existing rule. That is, we see if there is some feature \(X \sim c\) that makes Equation 3 perform better than Equation 2.

\[
\text{if } \text{diff.start} < -0.064 \& X \sim c \text{ then BUY else WAIT} \quad (3)
\]

If there is, we repeatedly add features that improve performance until no longer possible. The stopping rule is thus that we do not expand the tree if it does not lower the value of the loss function. After the stopping rule has been triggered, we gradually move back towards the root and seek to improve performance by expanding the tree from the leaves deciding `wait`. We do so not by expanding the existing rule, but by creating a new one.

In Equation 4, the root feature \(X_i \sim c_i\) was improved by the additional feature \(X_j < c_j\), but adding more features does not improve performance. We thus see if Equation 4 and 5 combined perform better than Equation 4 alone.

\[
\text{if } X_i \sim c_i \& X_j < c_j \text{ then BUY else WAIT} \quad (4)
\]

\[
\text{if } X_i \sim c_i \& X_j \geq c_j \& X_k \sim c_k \text{ then BUY else WAIT} \quad (5)
\]

In Equations 4 and 5, the first features are identical, and the second are complements. Adding a rule with only the two first features of Equation 5 would remove the effect of the second feature in Equation 4. In the additional rule, we have to include a third feature before we can evaluate the new rules’ effect on performance. Equation 4 alone and combined with Equation 5 are depicted as decision trees in Figure 3.
If performance improves after adding a new rule, we try to expand it by evaluating more features like when comparing Equation 2 and 3. Otherwise, we continue towards the root feature and evaluate more alternative rules at each leaf with response *wait*. We do so until all such leaves have been evaluated and when no further improvements can be made.

### 3.3 Benchmarks

To provide frames of reference to judge how well our algorithm performs, we compare the result to those of methods available to the ordinary customer. One such method is to always buy some number of days prior to departure. We choose the number of days that yields the lowest average price in the same manner as in Section 3.2. This measure could be misleading due to the occurrence of price spikes. Sometimes the price heavily increases only to drop to about the same level as before the following day. To avoid buying at such a peak, something the ordinary consumer probably would not do, we add the restriction that the *diff.previous* attribute may not exceed 0.5. If so, the purchase is made as soon as this attribute has an acceptable value. This yields that it is best to buy 94 days prior to departure.
We also see if our model performs better than choosing a day at random, which in the long run should yield the average price across all days of departure. Again, we want to exclude the price spikes, and do so by comparing our model to the median. From this calculation we exclude days closer than seven days to departure, as these were never available to any other method.
4 Results

In this section we present the final algorithm in the form of both a decision tree and a set of rules. We also present its overall performance against some benchmarks and show some examples on how it with varying success buys tickets. We then draw some conclusions and discuss how the method can be improved.

4.1 Final Tree and Algorithm

Applying the method described in Section 3.2 on our data yields the decision tree in Figure 4, from which we can derive two rules. Rule 1 is the set of features that matches the upper leaf with response buy, and rule 2 the lower. These rules, along with the last resort rule, constitute the algorithm that decides whether to purchase or not. It is presented as pseudo code in Table 3, where each rule is described as an if-statement.
if \( \text{diff.start} < -0.064 \) & \( \text{run.diff} < -0.077 \) & \( \text{days.depart} \leq 102 \) & \( \text{sigma} < 1.6 \)

or if \( \text{diff.start} \geq -0.064 \) & \( \text{days.depart} \leq 27 \) & \( \text{run.count} \leq 0 \) & \( \text{diff.start} < 0.17 \)

or if \( \text{days.depart} = 7 \)
then BUY
else WAIT

Table 3: The algorithm presented as pseudo code. The first, second and third if-statements represent rule 1, rule 2 and the last resort rule respectively.
4.2 Performance

When evaluating the algorithm on test data, it on average purchases tickets at a price of 9.5% above the lowest price. In 24 cases out of 122 it identifies the lowest price. The method of buying tickets 94 days prior to departure resulted in paying 18.6% above optimum. The median price is higher than both of these averages. These numbers are presented in Table 4.

In the test data, missing values make up about 9% of the observations, but about 16% of purchases are made the day after a missing value. We thus have an indication that the algorithm tends to react to missing values. Since missing values do not exist in reality, this could mean that the results are misleading.

4.3 Examples

In Figure 5, we see for what prices a purchase is possible with respect to the \textit{diff.start} and \textit{days.depart} attributes. They differ from the other attributes in that they have fixed reference points, which makes their effects easier to depict. With more than 102 days until departure, a purchase is not possible. Between 102 and 27 days, only rule 1 applies. With 27 days until departure or less, the looser rule 2 could also trigger a purchase. Note that the lower limit of the shaded area is set with respect to the first price observed and will differ between time series.

<table>
<thead>
<tr>
<th>Average Price</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.095</td>
<td>Our algorithm</td>
</tr>
<tr>
<td>1.186</td>
<td>Buying 94 days prior to departure</td>
</tr>
<tr>
<td>1.192</td>
<td>Median across all time series</td>
</tr>
<tr>
<td>1.000</td>
<td>Always buying the cheapest ticket</td>
</tr>
</tbody>
</table>

Table 4: Average price paid by different methods over all days of departure in test data. Although not an average, we include the median price for the sake of simplicity.
Figure 5: Fictive price fluctuation. Purchase not possible for points in shaded area due to `diff.start` and `days.depart` attributes.

In Figure 6, we see an example of when the algorithm found the lowest price. We see two clear declines in the price; one starting at about 85 days left until departure, and one at about 60 days until departure, at the bottom of which a purchase is triggered. The importance of good tuning of the thresholds is illustrated here. With looser restrictions on the `diff.start` variable, a purchase could have been triggered at the first decline resulting in a higher price paid. This also shows the importance of basing the purchase decision on more than one feature. The decrease during the first decline is larger than that in the latter one, but since the price was too high, the algorithm decided to wait.
Essentially, rule 1 finds clear price declines. In Figure 7, no decrease in the price is clear enough, leaving the purchase decision to rule 2. Furthermore, if we in this case were to add a shaded area like in Figure 5, we would see that rule 1 could never trigger a purchase because of the low initial observation. Rule 2 comes into play with 27 days until departure, but a purchase is triggered at 26. The algorithm decided to wait due to the `run.count` attribute, which in rule 2 makes sure that purchases are not made if the price is higher than the previous day.
As already presented, rule 2 does not trigger a purchase if the *diff.start* variable is larger than 0.17. Because of this and what has been explained about rule 1, the algorithm fails to purchase when no clear declines occur, and when at the same time the price increases quickly. In those cases, a purchase is triggered by the last resort rule. When evaluating the algorithm on test data, this happens four times out of 122.

In Figure 8, there is an early decline that rule 1 fails to recognize. Except for that, the price increases so much that with 27 days left until departure, rule 2 does not purchase either due to the high price. Hence, an expensive purchase is triggered with only seven days left until departure.

![Figure 8: Price fluctuation for flights departing 2017-07-10. Gaps in graph are due to missing values. Purchase triggered where the dashed lines intersect. The algorithm paid 1.24 with 7 days left until departure.](image)

### 4.4 Conclusion

Our algorithm clearly outperforms both benchmarks. As has been described, the method in this study is flawed in ways that harms the validity of the results. However, we do not deem the flaws to be severe enough for the results to be dismissed. Hence, our overall assessment is that an algorithm, based primarily on time series features selected by a decision tree, can make profitable recommendations about when to purchase a flight ticket.

### 4.5 Future Work

The main concern regarding the method has to do with the purely technical aspects of building the decision tree. There is no available R package or
script suited for this task and the scope of the thesis does not allow for an automatized procedure to be coded. The tree must thus in part be built "manually", in so far as that the code must be updated for every new combination of attribute and sign that is evaluated at every junction. This inconvenience makes it tedious and time-consuming to build the tree. A completely automatized procedure would in practice enable several things that could improve performance.

For instance, performance enhancing techniques like boosting could be used. In summary, boosting means creating several trees based on disjoint subsets of the training data. The decision is then made based on a vote among the trees (Hastie, Tibshirani and Friedman 2009, 337). It would also enable evaluating combinations of features at each junction, unlike only one as is currently the case. Both are currently possible in theory, but not in practice since the number of evaluations needed would increase exponentially.

In its current state, we deem the method robust against overfitting since it is hard to adapt the model to peculiarities in the training data when evaluating one continuous feature at a time, and since we are dealing with time series. By only evaluating one continuous feature at a time (some are discrete but are treated as continuous), decisions are based on some threshold with respect to some attribute at each junction. If we were to allow more than one feature, we could instead get that with respect to some attribute, all observations within some span are selected. This in turn would allow the tree to capture lows that the blunter current procedure misses out on. Overfitting is made even harder since we are dealing with time series and not cross-sectional data, because in the former, order matters. Some set of features that would trigger a purchase if presented the lowest point in a time series could also trigger a purchase at an earlier point. In other words, identifying the properties of the cheapest day is not necessarily enough, since the attributes could be similar or identical to an earlier day with a higher price.

If the building procedure were to be made more sophisticated, the same thing must be done to the stopping rule. As stated in Section 3.2, we do not add features to a rule if performance on training data does not improve. This naïve stopping rule is sufficient with a method robust to overfitting but must be made stricter as the risk of the same increases.
5 References


