Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Bachelor's Thesis Nr. 23b

Systems Group, Department of Computer Science, ETH Zurich

in collaboration with

Amadeus IT Group SA

Analysis and Prediction of Flight Prices

by

Jérémie Miserez

Supervised by

Prof. Donald Kossmann, Prof. Andreas Krause and Dr. Maria Grineva

March 2011 - September 2011

inf | Informatik
Computer Science

[page 1 (Abstract) redacted]

Acknowledgments

I would like to thank my supervisors Prof. Donald Kossman, Prof. Andreas Krause and Dr. Maria Grineva for their ideas, support and feedback as well as my project partner Florian Froese for his ideas and contributions to all implementations and evaluations.

Contents

Abstract **3**

Acknowledgments **3**

1 Introduction **5**
1.1 Objective . . . 5
1.2 Related work . . . 5

2 Exploratory data analysis **6**
2.1 Dataset overview . . . 6
2.1.1 Structure . . . 6
2.1.2 Definitions . . . 7
2.1.3 Extent of the dataset . . . 8
2.2 Distribution of values . . . 9
2.2.1 Interesting properties of the distribution of values . . . 11
2.3 Exploring price changes . . . 12
2.3.1 Patterns in price developments . . . 19

3 Prediction of airfares **21**
3.1 Classification methods . . . 21
3.1.1 General idea . . . 21
3.1.2 Online algorithms for classification . . . 21
3.1.3 Dataset generation . . . 26
3.1.4 Extensions to the basic algorithms . . . 26
3.2 Processing steps . . . 28
3.2.1 Preparation . . . 28
3.2.2 Labeling . . . 29
3.2.3 Feature vector generation . . . 30
3.2.4 Training and validation pipeline . . . 30
3.2.5 Baseline . . . 32

4 Experimental results **33**
4.1 Jobs run . . . 33
4.1.1 Data cleaning . . . 33
4.1.2 Datasets selected for training/validation . . . 33
4.1.3 Classifications run . . . 34
4.1.4 Parameters chosen . . . 34
4.2 Results . . . 36
4.2.1 Classification of current data . . . 36
4.2.2 Weight vectors for the classification of current data . . . 41
4.2.3 Prediction of future data . . . 48
4.2.4 Discussion . . . 51
4.2.5 Possible improvements . . . 55

5 Application: Web interface **56**
5.1 Features 56

Bibliography **59**

A Appendix **60**
A.1 Full description of feature vector 60
A.2 Additional images 63
A.3 Additional tables 65
A.4 Amadeus original record description 66
A.5 Source code & additional files 67

Chapter 1

Introduction

How airfares are calculated for each airline is not public knowledge. With each airline selling a myriad of different tickets and using proprietary rule sets to calculate the current prices, it becomes increasingly difficult to buy an affordable ticket. Being able to reliably predict the price of a certain ticket can help both businesses and private customers make informed decisions about what tickets to buy and when.
A large dataset with airfares from multiple airlines spanning several years was provided by Amadeus, which is one of the leading providers of IT services for the travel and tourism industry and operates the Amadeus global distribution system. Using this dataset, a classifier was built to distinguish cheap from expensive tickets. A web application then enables a potential customer to predict future price developments.

1.1 Objective

The main goals of this project were:

1. To analyze the patterns of price changes over time for individual routes.
2. To construct a general classifier that can be used for price prediction and which distinguishes between affordable and expensive tickets.
3. To predict future prices for tickets.
4. To analyze the factors that have an impact on the price.

While the large dataset provided ample opportunity to only analyze very specific subsets, the focus of this project was specifically on analyzing large parts of the dataset at once. Analyzing the patterns of price changes was done for single routes, while the classifier used large parts of the dataset to train its model parameters and make generalized predictions. The trained model could then again be analyzed to reveal general patterns.

1.2 Related work

Trying to predict the price of an airline ticket is not a new idea. However, the amount and quality of data to be processed significantly affects what algorithms can be employed.
The authors of [2] achieved good results by creating "buy" and "wait" rules for sequences of airfares for individual tickets over multiple weeks. A set of rules is created for each individual ticket and route, modelling the changes in price. The website Bing Travel [5] (formerly FareCast) currently uses this patented[1] approach for its airfare predictions.
However, for this project it was not desirable to use individual rule sets for each destination/ticket. Rather, a generalized classifier using large parts of the dataset at once was used.

[1] U.S. Patent No. 7'010'494

[pages 6-20 (Chapter 2) redacted]

Chapter 3

Prediction of airfares

The large dataset and the fact that there are no explicit links between records made it impossible to analyze price changes of an individual round-trip. It was much more practical to develop a model that generalized the properties of all records in the dataset. As discussed in Section 2.3.1, it should be possible to separate the few cheap records from the bulk of more expensive records and determine the properties that lead to the higher minimal prices as seen in Figure 2.14(b). Classification methods are able to separate records and determine the factors that lead to low or high prices.

3.1 Classification methods

3.1.1 General idea

In order to identify which records represent cheap tickets and which records have traits identifying them as expensive tickets, a classifier able to distinguish between "expensive" and "cheap" records is necessary.
It should be possible to train such a classifier on all records at once, identifying the features making a record cheaper or more expensive than other records. As some routes are more expensive than others, it does not make sense to include the route as a feature, but rather normalize prices per route. This enables comparison of prices across all routes without simply marking all records of a particular route as expensive. Each record is then labeled according to the normalized price. In short, a record for a particular route is labeled as "expensive" ($+1$) if its price is higher than the average price of all records for that specific route. Otherwise it is labeled as "cheap" (-1).
After training the classifier, it should be able to predict a label from a new record with an unknown price. As the route of the new record is known, a numerical minimal or maximal price (the aforementioned average price per route) can be directly inferred from the predicted label. Additionally, the model parameters of the trained classifier should contain information on how much each feature contributes to a record being cheap or expensive.

3.1.2 Online algorithms for classification

Due to the large amount of data, algorithms using more than a constant amount of memory are not suitable. Two algorithms were implemented, one for online support vector machines and the other for online l_1-regularized logistic regression. Both are convex optimization problems, which can be solved iteratively using the stochastic gradient descent (SGD) method. Furthermore, this method can be parallelized for use with large datasets. This allows efficient training of the classifier on a parallel system with limited memory, such as the Hadoop MapReduce cluster used for the initial data exploration.

Some definitions and terminology as they are used in the following sections:

x_i

Each data point x_i represents the features of a single record R_i and is also called the feature vector for record R_i. The contents are described in Section 3.2.3 and are derived from the fields of R_i, with the exception of route and price fields. Each component contains information about a single aspect of the record R_i.

y_i

Each label y_i represents the label (classification) of a single record R_i. Training data for a record R_i always consists of a pair (x_i, y_i) where both values are known. For new data points x_i the value of the labels y_i is initially unknown and is the result of the classification/prediction. A label has only two possible values: -1 and 1.

w

The weight vector w is the model parameter of the classifier to be estimated and is initially unknown. In both classifiers discussed below, w has the same number of dimensions as a data point x_i and determines the effect each value in x_i has on the classification result.

Stochastic gradient descent (SGD)
Given a convex set S and a convex function f, we can estimate the parameter w in $\min_{w \in S} f(w)$, where $f(w)$ is of the form $f(w) = \sum_{t=1}^{n} f_t(w)$. Usually, each summand f_t represents the loss function for a single observed data point from the dataset. Finding w is done iteratively, by using one random sample data point from the dataset per iteration. For regularization, $w \in S$ needs to be ensured, thus a projection onto S is necessary.
Let $w_0 \in S$ be the starting value. Then each iteration t consists of the update step

$$w_{t+1} = Proj_S(w_t - \eta_t \nabla f_t(w_t))$$

where $Proj_S$ is a projection onto the set S, η_t is the current step size (learning rate), and ∇f_t is the gradient of f approximated at the sample data point for iteration t. Each iteration the learning rate is set to $\eta_t = 1/\sqrt{t}$, ensuring that the parameter w is not only calculated from the data point seen in the latest iteration but from previous points as well.
It is possible to only use a subsample of the full dataset if the data points used for training are picked at random from the dataset. Training can then either be halted after a fixed number of iterations or as soon as sufficient accuracy is achieved.

Parallelized stochastic gradient descent (PSGD)
SGD can be parallelized[1] without much overhead. The data set is randomly partitioned and distributed to k machines, which each run SGD independently and produce a parameter w_i. After each machine has processed its data points, the arithmetic mean

$$w = \frac{1}{k} \sum_{i=1}^{k} w_i$$

produces the final result w. No intermediate variables need to be shared or synchronized while the machines are running SGD.

[1] see [7]

3.1.2.1 Online support vector machines with PSGD

A support vector machine (SVM) is a binary linear classifier. As such, a SVM takes a set of input data points and for each point predicts (decides) which of the two possible output classes the data point belongs to. In order to make correct classifications, a training algorithm using a labeled set of training data points with known classes is used to train the model. The model parameters represent the hyperplane separating the two classes from each other with a maximum margin. Figure 3.1 shows this graphically.

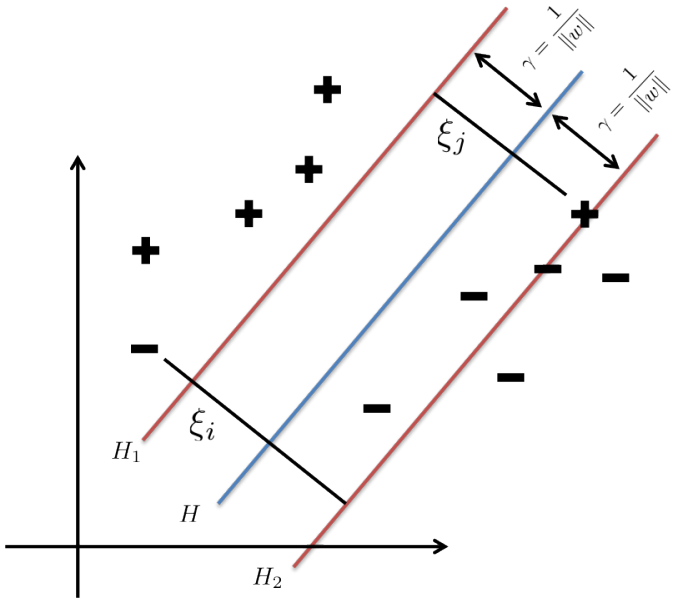

Figure 3.1: Graphical representation of a SVM with labels + and -, separating hyperplane H with margins γ, and slack variables ξ_i, ξ_j. Any point x_H, x_{H_1}, x_{H_2} lying on the corresponding hyperplane H, H_1 or H_2 satisfies the respective condition: $w^T x_H + b = 0$, $w^T x_{H_1} + b = 1$, $w^T x_{H_2} + b = -1$.

Let w, b be parameters of the separating hyperplane $w^T x + b$, γ the margin, x_i the i-th training data point, y_i the label of the i-th training data point. Then the following is a simple formulation of an SVM:

$$\max_{w,b,\gamma} \gamma, \text{ s.t. } y_i(w^T x_i + b) \geq 1 \text{ and } \gamma = \frac{1}{\|w\|_2}$$

After training, the decision rule

$$y = sign(w^T x + b)$$

classifies a new data point x as belonging to either the class + ($y = 1$) or - ($y = -1$).
Rewritten as a minimization:

$$\min_{w,b} w^T w, \text{ s.t. } y_i(w^T x_i + b) \geq 1 \text{ , with } w^T w = \|w\|_2^2$$

Added slack variables ξ_i:

$$\min_{w,b,\xi \geq 0} w^T w + C \sum_i \xi_i, \text{ s.t. } y_i(w^T x_i + b) \geq 1 - \xi_i$$

with

$$\xi_i = \begin{cases} 0 & \text{, if } y_i(w^T x_i + b) \geq 1 \text{ (decision correct, no penalty)} \\ 1 - y_i(w^T x_i + b) & \text{, if } y_i(w^T x_i + b) < 1 \end{cases}$$

account for non-separable training data points (on the "wrong" side of the hyperplane) and enable processing of datasets with noise. For the implementation, a formulation[2] with a hinge loss function (slack variables) and added regularization term with parameter λ is used. Additionally, b is set to $b = 0$ and all data points x_i are replaced by their component-wise normalized counterpart x_i'. In x_i', each component (feature) has a mean value of 0 and unit variance over all data points $x_{1,\ldots,n}$ in the dataset.
Such an SVM models a hyperplane passing through the origin, the optimization problem for n training data points is then[3]:

$$w^\star = \arg\min_w \lambda \|w\|_2^2 + \sum_{i=1}^{n} \max(0, 1 - y_i(w^T x_i'))$$

or equivalently,

$$w^\star = \arg\min_w \sum_{i=1}^{n} \max(0, 1 - y_i(w^T x_i')), \text{ s.t. } \|w\|_2^2 \leq \frac{1}{\lambda}$$

yielding the optimal model parameter w^*. Note that for the margin γ the following observation holds:

$$\gamma \geq \sqrt{\lambda}$$

as

$$\gamma = \frac{1}{\|w\|_2} \Rightarrow \|w\|_2 = \frac{1}{\gamma}, \text{ and } \|w\|_2^2 \leq \frac{1}{\lambda} \Rightarrow \frac{1}{\gamma^2} \leq \frac{1}{\lambda}, \text{ with } \gamma^2, \lambda \text{ positive}$$

When selecting a value for λ, care must be taken that the value is neither too large (risk of underfitting) nor too small (risk of overfitting) in order to achieve good generalization.
The model parameter w (weight vector) is then approximated using (P)SGD, with the respective terms being:

$$\text{Set } S = \left\{ w : \|w\|_2^2 \leq \frac{1}{\lambda} \right\}$$

$$Proj_S(w) = \begin{cases} \frac{w}{\lambda \sqrt{\|w\|_2^2}} & \text{, if } \|w\|_2^2 > \frac{1}{\lambda} \\ w & \text{, otherwise} \end{cases}$$

$$\nabla f_t(w) = \begin{cases} 0 & \text{, if } (w^T x_i) y_i \geq 1 \\ -y_i x_i & \text{, if } (w^T x_i) y_i < 1 \end{cases}$$

Accuracy of the trained classifier can then be tested by comparing classifications (using the decision rule) against the known labels of a second set of training data points.

[2] see [3], part 1.2
[3] see [4], slides 7, pg. 5

3.1.2.2 Online l_1-regularized logistic regression with PSGD

Using the same general idea of a hyperplane separating the data points of two classes, it is now possible to construct a linear classifier which not only classifies a data point but also returns the probability that a given classification is indeed correct[4].
To this end the logistic function is used. It is defined as

$$f(z) = \frac{1}{1 + e^{-z}}$$

and for an input z yields values from 0 to 0.5 for $z < 0$ and values from 0.5 to 1 for $z > 0$.
Again using the separating hyperplane $w^T x + b$ and the decision rule $y = sign(w^T x + b)$ logistic regression models the probability that x belongs to the class y as:

$$P(y|x, w, b) = \frac{1}{1 + e^{-y(w^T x + b)}}$$

The likelihood of some training data $D = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ is

$$L(D|w, b) = P(y_1, \ldots, y_n | x_1, \ldots, x_n, w, b) = \prod_{i=1}^{n} P(y_i|x_i, w, b)$$

Instead of maximizing the likelihood $L(D|w, b)$, the $\log$ of the likelihood can be maximized:

$$w^\star, b^\star = \arg\max_{w,b} \sum_{i=1}^{n} \log(P(y_i|x_i, w, b))$$

Rewritten as a minimization of $\log(\frac{1}{L(D|w,b)}) = -\log(L(D|w, b))$:

$$w^\star, b^\star = \arg\min_{w,b} \sum_{i=1}^{n} \text{logloss}(y_i, x_i; w, b)$$

with

$$\text{logloss}(y_i, x_i; w, b) = -\log(P(y_i|x_i, w, b)) = -log(\frac{1}{1 + e^{-y(w^T x + b)}}) = \log(1 + e^{-y(w^T x + b)})$$

For the implementation a regularization term with parameter λ is added. As in the online SVM, b is again set to $b = 0$ and the data points x_i' instead of x_i are used. The optimization problem for n training data points is then[5]:

$$w^\star = \arg\min_{w} \lambda \|w\|_1 + \sum_{i=1}^{n} \text{logloss}(y_i, x_i'; w, 0)$$

or equivalently,

$$w^\star = \arg\min_{w} \sum_{i=1}^{n} \text{logloss}(y_i, x_i'; w, 0), \text{ s.t. } \|w\|_1 \leq \frac{1}{\lambda}$$

This classifier has two distinct advantages over the online SVM. Using l_1 regularization should result in sparse solutions (feature selection) for w while $P(y|x, w, 0)$ gives a confidence measure for the classification.
Choosing a value for λ has similar effects as for the online SVM, with large values bearing the risk of underfitting (more features will be 0) and small values leading to overfitting.
The model parameter w ("weight vector") is approximated using (P)SGD, with the respective terms being:

[4] see [3], part 2
[5] see [4], slides 7

$$\text{Set } S = \left\{ w : \|w\|_1 \leq \frac{1}{\lambda} \right\}$$

$$Proj_S(w) = \begin{cases} \text{L1PROJECT} & \text{, if } \|w\|_1 > \frac{1}{\lambda} \\ w & \text{, otherwise} \end{cases}$$

$$\nabla f_t(w) = -\frac{x_i y_i}{e^{y(w^T x)+1}}$$

where L1PROJECT is the linear-time projection algorithm onto the l_1 ball as described in Chapter 4 of [1].

3.1.3 Dataset generation

In order to train a classifier, a training dataset containing records with known prices is necessary. For each record R_i in such a training set R, a normalized feature vector x'_i and a label y_i needs to be generated. A few definitions are needed:

μ_r
: The arithmetic mean of the price over all records in the dataset with route $= r$

σ_r
: The standard deviation of the price over all records in the dataset with route $= r$

p'_i
: Each normalized price p'_i represents the normalized price of a single record R_i. Normalized prices are calculated as the standard score $p'_i = \frac{p_i - \mu_r}{\sigma_r}$, with p_i being the original numerical price of a record. Note that the values of μ_r and σ_r are dependent on the route r of the particular record R_i.

The label of a record in the training set is then:

$$y_i = sign(p'_i)$$

For each record, a feature vector x_i is created. After the creation of all N feature vectors $x_{1,\ldots,n}$, each feature is normalized (standard score) across all vectors. For each feature this results in a mean of 0 and unit variance over all vectors $x'_{1,\ldots,n}$ in the training set.
In order to predict future prices (labels), a prediction dataset only needs the normalized feature vectors x'_i for each record and a trained classifier with weight vector w

3.1.4 Extensions to the basic algorithms

The classifiers introduced only allow for classification into one of two classes, i.e. every point is treated the same. This might not be the best solution to the real-world problem of predicting a price.

3.1.4.1 Multiclass classification

It is desirable to have more than two possible output classes, where each class represents a smaller price range. To this end, a recursive classification method is used, with several separately trained classifiers in a binary tree arrangement. Figure 3.2 shows this graphically for a total of 8 output classes.
The total price range is split up into 8 classes, each representing a range of size 0.5σ, with class 7 having a lower bound of 0 and class 14 having the upper bound of $+\infty$. Each classification into one of the 8 classes also classifies the record into 2 of the larger price ranges. As each classifier has a classification error, the probability of error is compounded with each level. Therefore, there is a trade-off between small price ranges and low error probabilities. Note that this recursive approach is only possible because the price ranges are guaranteed not to overlap, otherwise classifications would be ambiguous.
As showed in Section 2.3.1, the price distribution generally represents a normal distribution, at least for large enough datasets. With a normal distribution of prices it is clear that more records should fall into

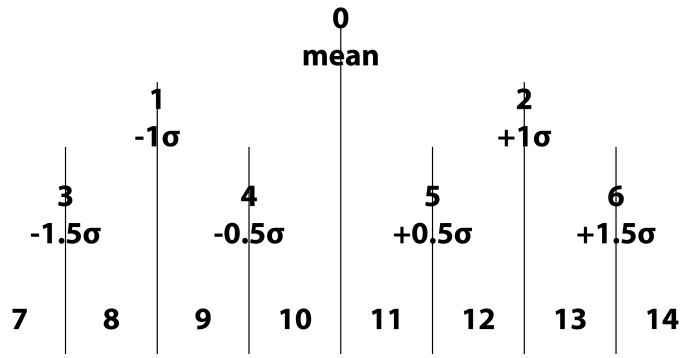

Figure 3.2: The 7 classifiers (numbered 0-6) are depicted as lines, the 8 possible output classes (numbered 7-14) representing price ranges fill the space between two lines.

the innermost classes 9-12 ($\pm 1\sigma$) than in the outer classes 7,8 and 13,14. The price ranges chosen should allow interesting prices (such as very cheap or very expensive ones) to be more accurately separated from the bulk of average prices.
Each classification now uses 3 classifiers generating a total of 3 labels for each record instead of 1, while classifying data into 8 (2^3) price ranges instead of 2.
Multiple labels y'_i,y''_i,y'''_i are now needed, one for each classifier:

$$y'_i = sign(p'_i - d'_i) \quad y''_i = sign(p'_i - d''_i) \quad y'''_i = sign(p'_i - d'''_i)$$

with d'_i,d''_i,d'''_i being the decision boundaries of the first, second and third classifiers for the record R_i.

3.1.4.2 Per-airline classification

Some fields of a record can have different interpretations with different airlines. Three examples are the flight numbers, the booking classes and the availabilities. In order to account for this, a separate classifier is used for each airline. While this increases the number of classifiers overall, it does not increase the number of classifiers used on each record. A negative side effect is that it significantly decreases the number of records each classifier has available for training. To counteract this side effect, only the top 20 airlines use separate classifiers while all other airlines are grouped together into a 21st set of classifiers. As discussed in Section 2.2.1, 84.53% of all records with an airline have one of the top 20 airlines listed as their airline.
In order to compare per-airline classification with global classification, an additional classifier which has all records available for training is needed. This doubles the total number of classifiers used on each record. The number of weight vectors that need to be stored on disk is increased from 7 when using global multiclass classification to $7+7 \cdot 21 = 154$ when using multiclass per-airline classification. Combined with multiclass classification, each record runs through a total of 6 classifiers (3 for global classification, 3 for per-airline classification) generating a total of 6 labels:

$$\begin{array}{lll} y'_{\text{i, global}} = sign(p'_i - d'_{\text{i, global}}) & y''_{\text{i, gl.}} = sign(p'_i - d''_{\text{i, gl.}}) & y'''_{\text{i, gl.}} = sign(p'_i - d'''_{\text{i, gl.}}) \\ y'_{\text{i, airline}} = sign(p'_i - d'_{\text{i, airline}}) & y''_{\text{i, airl.}} = sign(p'_i - d''_{\text{i, airl.}}) & y'''_{\text{i, airl.}} = sign(p'_i - d'''_{\text{i, airl.}}) \end{array}$$

3.2 Processing steps

3.2.1 Preparation

For the implementation, a number of practical issues needed to be resolved, as the records received from Amadeus were not immediately usable for training a classifier. The full dataset files were organized by route into folders for each From-Airport and subfolders for each To-Airport. Each individual file was already sorted by request date. Because the route could be inferred from the folder structure, both the From-Airport and To-Airport values of a record were always available.

3.2.1.1 Data cleaning

All records were inspected for errors and impossible values. A record was deemed invalid and discarded if it fulfilled at least one of the following criteria:

- Is an incorrectly formatted record
- Contains unusable numerical values such as infinity or NaN
- Contains unusable date values
- Contains unusable/impossible categorical values
- Has a total price below 0
- Has a number of passengers ≤ 0
- The number of hops is ≤ 0
- Not all inbound sequences have the same length
- Not all outbound sequences have the same length
- A field necessary for training or validation is empty:
  - Tax, Fare, Currency
  - Number of passengers
  - Each entry in every sequence of cabin classes

While most checks were automated, determining the unusable/impossible categorical values involved creating a list of all categorical values present and manually whitelisting only the usable categories. Usually, unusable categorical values were the result of a misaligned entry and easily detectable, e.g a date value as a cabin class. This approach limits the handling of new data, as new categories must be manually added to the whitelist.

3.2.1.2 Data selection

Instead of using all records available for training of the classifier, multiple smaller datasets were generated. These were generated by a combination of subsampling and filtering by routes, cabin classes and request data ranges. When filtering by cabin class, there are different possibilities to handle a record with different cabin classes on each hop. It is not clear which of the hops is the longest or which has the most influence on prices. Therefore, in this implementation only records with identical cabin classes on all hops were considered when filtering by cabin classes, while all records with mixed cabin classes were not selected.

3.2.1.3 Partitioning & Shuffling

All records were originally stored in files sorted by request date. In order to ensure that the order of processing has as little effect as possible, the dataset was randomly shuffled. Parallelization using PSGD requires the dataset to be split into multiple parts or a larger number of small chunks, so that each machine can run the classification algorithm on their assigned chunks.
Using MapReduce, shuffling and partitioning was done as follows:

- Map step: Create key-value pairs:
  - Key: A random 64bit integer.
  - Value: The record itself
- Reduce step: Write all values to disk. Note that the number of reducers determines the number of files written. Each file represents a chunk.

Assuming no key is used multiple times, this algorithm produces a random permutation of all records. As the total number of records available is considerably smaller than the range of a 64bit integer, the probability of a key being used multiple times when shuffling is very small.

3.2.2 Labeling

For validation and training, each record was labeled according to the normalized price. Each record received 6 labels in total, each having either a value of -1 or 1. The corner case of the sign function returning 0 were interpreted as a label of -1, because labels of 0 were not practical. Due to the usage of normalized prices, this corner case does not occur often.

3.2.2.1 Price normalization

Currency conversion
Currencies were converted to EUR on-the-fly analogous to the conversion used for the data exploration. Because conversion rates fluctuate, prediction of future data may not be accurate without manually adjusting the conversion rate table to incorporate the latest rates.

Computing the normalized price
Price normalization was handled in two steps. In a first MapReduce job, the mean price μ_r and σ_r for each route r were calculated. For a single route r, the following well-known formulas were used:

$$E[p] = \frac{1}{n}\sum_{i=1}^{n} p_i$$

$$E[p^2] = \frac{1}{n}\sum_{i=1}^{n} p_i^2$$

$$\mu_r = E[p]$$

$$\sigma_r = \sqrt{E[p^2] - E[p]^2}$$

where both sums were accumulated separately using the Kahan summation algorithm for precision. The values were stored on disk for each route r.
All following MapReduce jobs loaded the corresponding μ_r and σ_r from disk and computed the normalized price $p_i' = \frac{p_i - \mu_r}{\sigma_r}$ on-the-fly whenever necessary. This removed the need to write the normalized prices to disk.

3.2.3 Feature vector generation

For each record, a feature vector consisting of 930 features was created. Before normalization, each entry was set to either 1.0 (boolean true), 0 (boolean false) or a value associated with a numerical field in the record. The feature vector represents each record as a 930-dimensional vector.
A detailed listing of each feature can be found in Appendix A.1, while the following tables list the feature categories and some examples for each category.

Category	Used record fields
Dates	Request Date, Departure Date, Return Date
Date differences	Return-Departure Date, Departure-Request Date
Categorical values	Currency, Passenger Type, Airline
Numerical values	Number of passengers, Number of hops
Sequences of categorical values	Cabin Classes, Booking Classes, Availabilities
Sequences of numerical values	Flight numbers

Category	Example features
Dates	isMonday, isWeekend, isWinter, weekOfYear, ...
Date differences	containsFridayAndSaturday, isShorterThanOrEqual3, ...
Categorical values	currencyIsCHF, airlineIsLH, ...
Numerical values	isNrOfHops2, nrOfHopsAsNumericalValue, ...
Sequences of categorical values	cabinClassesContainsM, cabinClassesEndsWithM, ...
Sequences of numerical values	containsNrBetween1000And1999, ...

3.2.3.1 Feature vector normalization

As with price normalization, each of the 930 features f_m was normalized in two steps. In a first MapReduce job, the arithmetic means μ_{f_m} and standard deviations σ_{f_m} were calculated using the same methods as for price normalization and subsequently stored to disk. All following MapReduce jobs loaded the 930 means and standard deviations from disk and calculated the normalized feature vector x_i' on-the-fly by calculating the standard score of each feature f_m:

$$f'_m = \frac{f_m - \mu_{f_m}}{\sigma_{f_m}},\ m \in 1, \ldots, 930$$

3.2.4 Training and validation pipeline

Figure 3.3 shows a general outline of all the steps that were necessary to obtain the final weight vector w_{mean}. Each of the k MapReduce training jobs produced an intermediate weight vector w_k from which the averaged vector w_{mean} was calculated in another MapReduce job. The validation MapReduce job used an additional chunk to determine the accuracy of a classifier using the current w_{mean} as its weight vector. For each iteration, the chunks were randomly selected from a pool of all chunks that had not yet been used for training in an earlier iteration. In the implementation, new training jobs were already started while the last validation job was still running, thus speeding up processing of large datasets by a factor of ≈ 2.

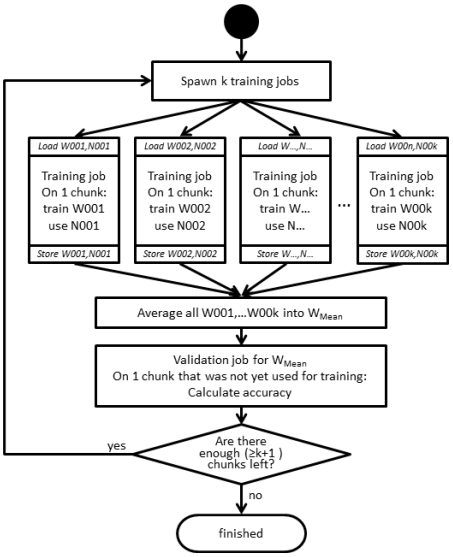

Figure 3.3: General outline of the training and validation steps.

For i iterations, the algorithm uses total of $k \cdot i + 1$ chunks, thus ideally only 1 chunk is not used for training after the algorithm exits. This algorithm is parallelizable to multiple machines, with the only limiting factor being the number of chunks in a dataset.

3.2.4.1 Training jobs

Each of the k MapReduce training jobs loaded the current weight vector w_k and record counter n_k from disk before it started to process any records. After the classifiers were trained using all records in the chunk, w_k and n_k were written back to disk for use with the next iteration. No record was used for training more than once and no data was shared between different jobs.

3.2.4.2 Averaging jobs

The averaging MapReduce job computed the arithmetic mean w_{mean} of all weight vectors w_k resulting from the latest iteration. No weighting was performed.

3.2.4.3 Validation jobs

Each validation MapReduce job used the latest weight vector w_{mean} to classify all the data points in a chunk. Accuracies were calculated as:

$$acc_{global} = \frac{\#\text{ of correct classifications using global } w_{\text{mean}}}{\#\text{ of records}}$$

$$acc_{airline} = \frac{\#\text{ of correct classifications using correct airline } w_{\text{mean}} \text{ for each record}}{\#\text{ of records}}$$

Records were only considered correctly classified if all the output class was correct, i.e. all 3 classifications in the multiclass hierarchy were correct. In addition, accuracies were calculated for each of the classifiers individually for comparison.

3.2.5 Baseline

In order to assess how good the obtained accuracies were compared to a perfect classification, an initial baseline MapReduce job determined the actual percentages of records per dataset in each price range. If the records in a dataset are unevenly distributed among the price ranges, the accuracy of a classifier can be high without the need for a particularly good classifier.
As an example, a classifier with two price ranges can be considered, with one range containing 90% of the records. A random classifier could now classify 50% of all input records to each range. The overall accuracy of this random classifier would be

$$acc_{random} = 0.5 \cdot 0.9 + 0.5 \cdot 0.1 = 0.5$$

Using a biased classifier which classifies all records to only one of the two price ranges a much higher accuracy of

$$acc_{biased} = 1.0 \cdot 0.9 + 0 \cdot 0.1 = 0.9$$

could be achieved without the need to even inspect each record.

3.2.5.1 Lambda selection

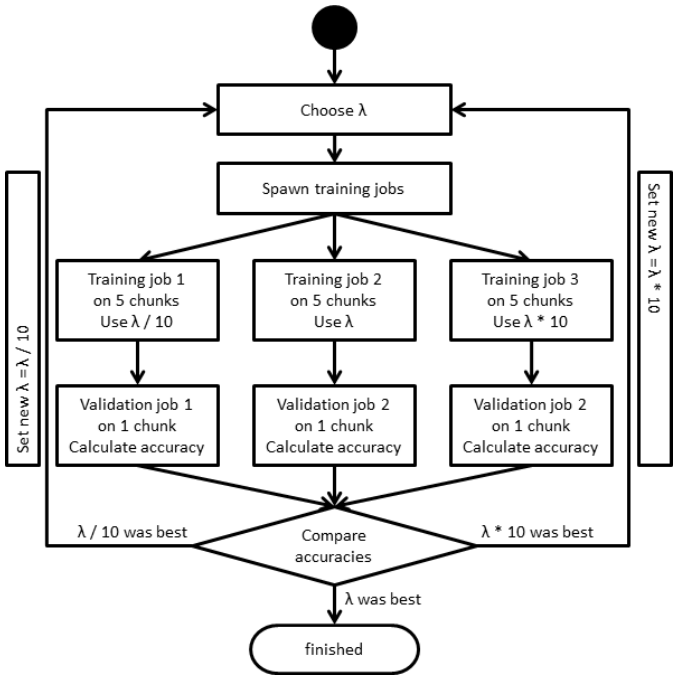

Figure 3.4: General outline of the lambda selection steps.

The algorithm as depicted in Figure 3.4 shows how a value for λ was chosen. Starting from an initial value of λ, the algorithm searched for the correct order of magnitude for the final λ. Five randomly selected chunks of the dataset were used for each training step, while the validation step used an additional chunk to calculate an accuracy value. Because both the training and validation steps are time-consuming, no search beyond the determining of the order of magnitude was done and regardless of the number of chunks in the dataset, always exactly 5 chunks were used for training. Therefore, each λ value found by this algorithm was only an approximation of the true ideal λ. This algorithm is parallelizable to 3 machines and does not make use of any pipelining for training and validation jobs.

[pages 33-55 (Chapter 4) redacted]

Chapter 5

Application: Web interface

The classification results in section 4 were generated in batches and classified only data belonging to the Amadeus dataset. For an end user interested in predicting a price, this is not very useful. To enable an unskilled user to predict a price, an interactive web interface was developed. The web application uses an already trained classifier and produces predictions for single and partial records as well as price developments. Results are calculated instantly, as there is no need for a MapReduce job when only predicting the price of one record.

5.1 Features

Data sources
Weight vectors are loaded directly from a remote Hadoop cluster running the training algorithms. This enables the web application to always use the latest up-to-date weight vectors even before all training iterations are finished running. The dataset to be used can be selected by using a simple dropdown menu. Alternatively, weight vectors can be loaded from a local file stored on the webserver. Figure 5.1 shows the settings page used to configure the data source.

Predicting a price
The form shown in Figure 5.2 lets the user input all values currently known. When some fields are left blank, the web application recognizes that only a partial record was entered. The corresponding missing features in the feature vector are then set to a neutral value of 0 (mean of the normalized feature vector), as not to influence the prediction. The form outputs not only the predicted class, but also the price ranges of all intermediate classifications and their general accuracy for both SVM and L1 classifiers and the internal classifier confidences for all L1 classifiers. The number of chunks the classifiers were trained on is listed as the number of iterations done.

Predicting a price over time
Figure 5.3 shows an example generated plot after prediction. Each predicted price range is drawn with upper and lower bounds, with the innermost range representing the final prediction. This representation allows for a faithful visualization of the classification process.

[pages 57-58 redacted]

Bibliography

[1] J. Duchi, S Shalev-Shwartz, Y. Singer, and T. Chandra. Efficient projections onto the l_1-ball for learning in high dimensions. *Proceedings of the 25th International Conference on Machine Learning*, 2008.

[2] O. Etzioni, C.A. Knoblock, R Tuchinda, and A. Yates. To buy or not to buy: Mining airfare data to minimize ticket purchase price. *SIGKDD*, 2003.

[3] Prof. Dr. Andreas Krause. Data mining: Learning from large data sets (lecture exercise no. 3), 2011. University lecture exercise; accessed online July 1, 2011; http://las.ethz.ch/courses/datamining-s11/hw/hw3.pdf.

[4] Prof. Dr. Andreas Krause. Data mining: Learning from large data sets (lecture slides), 2011. University lecture; accessed online July 31, 2011; http://las.ethz.ch/courses/datamining-s11/.

[5] Microsoft. Bing travel (website, formerly farecast). accessed online July 31, 2011; http://www.bing.com/travel/.

[6] . OANDA-Corporation. Historical exchange rates (website). accessed online July 31, 2011; http://www.oanda.com/currency/historical-rates/.

[7] M. Zinkevich, M. Weimer, A. Smola, and L. Li. Parallelized stochastic gradient descent. *24th Annual Conference on Neural Information Processing Systems*, 2010.

[pages 60-67 (Appendix) redacted]