CHALMERS |  UNIVERSITY OF GOTHENBURG

Scalable Machine Learning for Big Data

Bachelor's Thesis in Computer Science

EMANUEL ANDERSSON
EMIL BOGREN
FREDRIK BREDMAR

Department of Computer Science & Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2014
Bachelor's Thesis DATX02-14-02

Abstract

We describe each step along the way to create a scalable machine learning system suitable to process large quantities of data. The techniques described in the report will aid in creating value from a dataset in a scalable fashion while still being accessible to non-specialized computer scientists and computer enthusiasts. Common challenges in the task will be explored and discussed with varying depth. A few areas in machine learning will get particular focus and will be demonstrated with a supplied case-study using weather data courtesy of the Swedish Meteorological and Hydrological Institute.

Acknowledgements

We would like to extend our utmost gratitude towards our supervisor Prof. Laura Kovacs. Her guidance and motivation has been invaluable.

Thanks to all our professors and staff over the years at Chalmers and the University of Gothenburg.

The Authors - Sweden, Spring 2014

Contents

1

Introduction

You know that great feeling of not having to browse through all spam emails when checking your inbox? Or getting discounts on items at the local grocery store that are relevant to you? Personal recommendation lets you explore new artists at Spotify or unknown movies at Netflix. All those things are accomplished through machine learning. It is by modeling historical data and making statistical decisions based on the models that our inboxes are not flooded or the grocery store actually gives discounts on groceries you would buy. Machine learning can not solve every problem, but it has become a tool for solving problems previously hard to solve. Problems that are much too complex for people to handle with traditional methods, such as static condition rules. In this report we will explain step-by-step and discuss how you can take your data and your unanswered questions and produce new value and results.

Machine learning can be used as a tool to create value and insight which helps organizations to reach new goals. As we mentioned above in the examples for Spotify and Netflix, it is essential that their customers get relevant suggestions. Throughout the report, uses for the case-study based on the data from SMHI (Swedish Meteorological and Hydrological Institute) will be explored, ultimately answering three questions: *Is there going to be storm? How will the weather be in the future?* and *How will the weather be in a new city with no prior data?*

The challenge of scalable machine learning can be broken down into three different areas. First, you have to know your dataset and explore it to find questions you want answered and the format you can answer them with. Secondly, a scalable architecture which allows for cost-efficient computing has to be setup and configured. Lastly, you apply the right machine learning algorithms within the architecture on the dataset in order to produce valuable results. The difficulty of each step may vary depending on the sort of problem you want to solve. For example, exploring- and cleaning the dataset

might prove harder than the actual machine learning in some cases and vice versa.

Each section of the report can be read separately. Each section starts with an introduction and a brief description of the structure.

1.1 Dataset

Knowing your dataset will not yield any results in itself, but ignorance might be costly later on. Exploring your dataset helps you discover new areas of advance and reduce uncertainty, even eliminating unreachable goals early on. There are a few tools and tricks to handle common challenges with the dataset which are useful to master. Knowledge about missing values is an important tool for preprocessing data and also for the final results. For instance, all missing- and faulty values could be removed before processing the data.

Example: Reason about missing values.

1.2 Scalability

Scalability is essential when the dataset stops fitting on a single machine and the processing becomes unacceptably slow. If you plan ahead for your processing needs, much hassle can be avoided. For example, all scalable solutions should be implemented for at least two machines. That way most of the scaling problems are solved in the initial implementation if more processing power is needed. There exist numerous free- and proprietary solutions to assist with large scale processing. Each with its own pros and cons. Apache Spark[1] is one such solution and the solution that is used in this paper's case-study.

Example: Distribute processing to several computers.

1.3 Machine learning

Machine learning is about turning a question and some data into value by using statistical inference. Different algorithms suit different questions, and there are always important factors to consider. Three of the most common techniques are called classification, regression analysis, and recommender systems.

Example: Apply a classification algorithm to fight spam.

1.4 Case-study

A use case based on SMHI weather data[2] will be used as an example for the techniques described in this paper. The data can be used in several different ways and is in a form that is easy to understand and work with. It provides enough data so that a wide range of techniques can be applied to it.

Our report tackles the challenge of machine learning in three steps: dataset (chapter 2), scalability (chapter 3), and machine learning (chapter 4). Each step has its own chapter which deals with problems related to that step. The case-study will be used as examples throughout the text with results at the end. Lastly, there is a discussion section which expands on some topics encountered in the text.

2

Dataset

The dataset is the core of machine learning. It is the dataset that creates the opportunity to make heavy statistical computations to answer hard questions which are unfit to be solved with common techniques. The dataset is a gathering of information. A few examples of datasets are records of public transport, a grocery store's information about product sales during a period, stock prices, weather metrics, visitors to a website. The list can be made very long as there are no real constrains of what constitutes a dataset. However, to benefit from the information hidden in the dataset it is important to make sure to understand the data, and how it relates to the questions that are being asked. Without a scientific approach to the dataset, pitfalls such as implying causation just with the motivation of correlation[3] might lead to faulty conclusions.

This chapter will be tackled in three steps beginning with cleaning followed by exploration and then visualization. Each step is designed to increase the quality of the dataset as well as the understanding of the data and its properties.

2.1 Cleaning

Incomplete datasets, whether values are missing or faulty, are a big concern when working with tough problems (such as machine learning). There is no single perfect solution to fix an incomplete dataset. A lot of papers have been written targeting these issues. We will scratch the surface of this issue in this report. Hopefully it will be enough to see what type of problems your data may have.

2.1.1 Missing values

Having missing values in a dataset is more common than not. Extensive research has been done in the area because of this. This report will touch briefly on the subject. The

goal is to give a guiding hand toward addressing the structure and possible methods of correction for missing values.

The reason a dataset misses values is usually tied to the type of the dataset. The key to handling this issue is to look at the possible structure of the missing values. A broad categorization is to determine if your datasets is missing values completely random, missing values at random or non ignorable missing values. Missing completely at random is, like it sounds, that the values that are missing is not grouped in any way. Not in time, nor in group of similar other values. This is quite rare but if it occurs, there are some simple ways to create a value for the missing positions. Missing at random is a more realistic assumption.
Edgar Acuna et al. [**?**] discusses four ways missing values are usually handled:

Structure within missing values When visualizing the data, it is helpful to search for a structure among the missing values. Are they in clusters? Are they completely random? Or do they depend on other features? A simple and efficient categorization to determine the structure of missing values is to check which of the following categories the missing values belong to.

Missing completely random If the missing values in the dataset are completely random there is no way to see when or why they are missing. This is quite a rare case and nice to have when it comes to correcting or modifying the missing values.

Missing at Random Missing values at random is much more common than completely at random. When missing values at random it may be possible to see a reason for why groups of values might be missing, however, not all missing values follow the same condition to why it is not present. There is also no clear correlation between a missing value and the values indicating a missing value.

Not missing at random This type of structure is the hardest to handle. Here, there exists a very concrete relation between other values and the missing values. The reason this becomes difficult to handle is that the number of different techniques for correcting the missing values decreases rapidly. This is due to the low probability of similar data points to those which are missing.

Based on the above four approaches to handling missing data, several strategies have been developed. Which one to use depends on the dataset and the structure of the missing values. The most widely used strategy is called *Case Deletion*. This strategy simply removes all samples with missing values in them and use the new dataset without the missing values. It works well if the samples in the dataset are independent and the new dataset is still large. It is trivial to handle if the missing values are less than 1% of the total dataset. 1-5% are usually manageable.[5]

Mean Imputation is another strategy for handling missing values. Each missing value gets represented by the mean of that feature through the whole dataset. The benefit of this strategy compared to case deletion is that we get to keep the samples containing

missing values. However, it is important to be aware of the structure of the data points. If the spread of the values for computing the mean is large, then it can be dangerous to use this strategy, especially if the values are continuous in nature.

K-nearest neighbor imputation is a strategy where the k most similar data points are used to determine the missing value of a data point. This method is the most complex of the strategies we discuss in this report. The trade-off for this complexity is that the determined value often is close to the real value. This strategy is well-suited for datasets with a structure of completely random missing values since the probability of finding other samples with similar values in other features is high. It might work with datasets that have the structure of missing at random values. If the dataset have a structure of not missing at random, the probability of being able to use the KNN imputation strategy with good results is low. The way to implement this strategy is in itself a machine learning algorithm. It will be covered more in detail in the recommendation part of the machine learning chapter.

2.1.2 Faulty values

Faulty values are almost by definition misinformation and will only corrupt the final result. The exception is if the faulty values are by themselves the data being explored. An example of a faulty value is a misreading of a thermostat. Faulty values will usually have to be removed from the dataset.

2.2 Exploration

Often the dataset is accessible as files and can be explored with commands in the terminal. Storage systems, such as a database, usually offer similar tools for exploration. After navigating to the directory of the dataset in the terminal, usually by using the command *cd*, commands can be executed by typing the command followed by a number of arguments. *cat* prints the contents of a file to the terminal.

```
cat filename
```

Using globbing we can print all files ending with .txt. By using pipes we can then forward the result to another command. *grep* finds patterns in text and *wc* together with the argument $-l$ count the lines.

```
cat *.txt | grep 999.0 | wc -l
```

This command counts the number of missing or faulty values in the SMHI dataset. Commands such as *mv*, *cp*, *rm*, and *ls* are useful when managing files in the dataset.

Another way of exploring the dataset is to use a programming language. Code written for this purpose is usually reusable later on when writing the actual implementation. By asking simple questions, it is possible to get an idea of extensive missing values and discover information that is was not known on forehand. Some questions that could be asked about the SMHI dataset: *What is the highest temperature value? What is the mean precipitation?* and *How many cities have full coverage of data?*

2.3 Visualization

Exploring the shape, patterns, and characteristics of your data increase both your own understanding as well as your collaborators' understanding. It is the most important tool for exploration and often a product in itself. It can reveal characteristics which increase your accuracy later on and help you avoid pitfalls. A famous illustration of how seemingly similar datasets might have vastly different properties is Anscombe's quartet[6] shown in Figure 2.1. It is for example crucial to have a visual grasp of the dataset when picking the kernel for SVM[7].

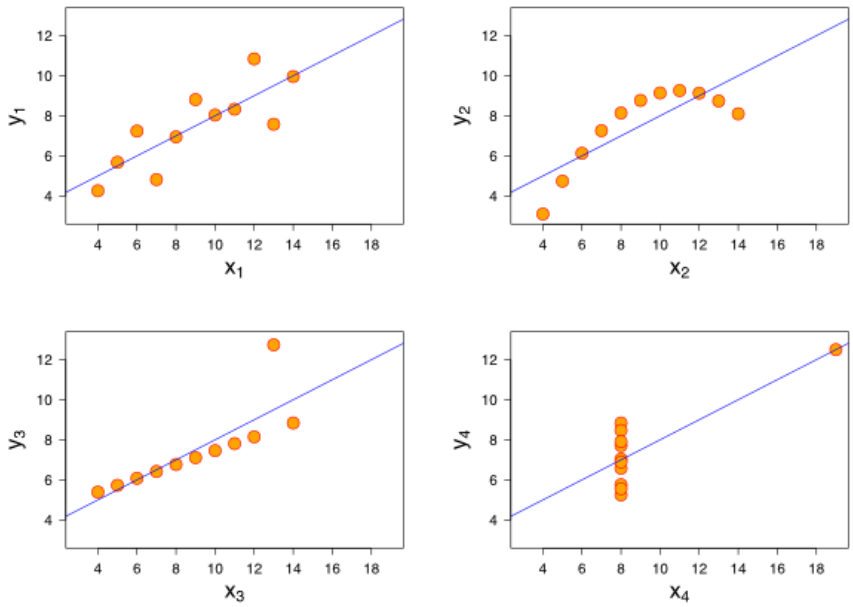

Figure 2.1: Anscombe's quartet

The ability to efficiently process the data on a single machine decreases as the dataset grows larger. Most datasets can be visualized by looking at key values but sometimes

that is not enough, and sometimes the dataset is too big. There are essentially two ways of handling such scenarios: either **sample the data or stream the data**. Streaming is beyond the scope of this paper as it combines elements from infrastructure and data-flow, however, most processing frameworks have tools for it. In our work we focused on the method of sampling the data, by picking the data at random in order to avoid bias. Using randomization when sampling data is a common practice in order to avoid so called biased samples, in other words, samples which are unfairly picked and provide a skewed picture of the population. Numerous tools exist to aid with visualization. Some software suits include visualization tools built-in, e.g. Microsoft Excel. Matplotlib[8] is a widely used visualization library for Python that is easy to setup and use. It also provides sufficient functionality for the advanced user. Figure 2.2 shows where SMHI's complete (1961-2011) weather stations are located. The stations are spread throughout the country quite evenly. It would probably be better to have some additional stations in the central-north areas.

Figure 2.2: SMHI's weather stations.

Figure 2.3 is a favorite and taken from visualization of our dataset. The figure shows mean temperature for each year between 1961 to 1996. There is a huge variation and nothing can really be concluded from this visualization.

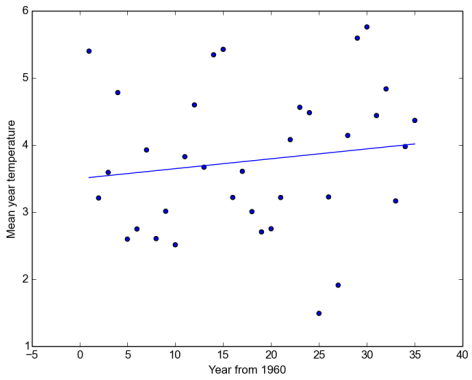

Figure 2.3: An example of the risk of abstraction and high variation

3

Scalability

Some tasks are too time consuming and computationally hard to be run on a single computer. Machine learning on large datasets is one such task. Using scaling techniques, even extensively heavy computations are possible. Scaling is the techniques used to handle increasing workload in a sustainable fashion, to expand resources that accommodate computation. There are two common approaches to scaling, vertical- and horizontal scaling. When scaling vertically, it is common to have one really fast and expensive computer to do all the heavy computation. Vertical scaling used to be more common back when commodity hardware was significantly slower than high priced hardware. Commodity hardware has gained a lot of performance and putting several computer together may yield a far superior system in most cases. Third-party companies offer hourly rental of hardware which have made horizontal scaling very attractive lately. This report will put more focus towards horizontal scaling since this is more commonly used by organizations today and is is more sustainable. There is a lot of buzz around big data and scaling but it is wise to question if the computations and dataset is heavy and large enough to actually benefit from scaling. If the speed of the computation is less important and the size of the dataset and computations are manageable on a single computer, consider running the program on a single machine. It is incomparably cheaper and easier to not have to scale. Do not scale just for the sake of scaling.

This chapter will contain a part about the concept of scaling followed by several frameworks and tools used to solve the task. We conclude this chapter with a brief discussion on clusters and fault tolerance.

3.1 Horizontal scaling

Vertical scaling is about increasing processing power to existing machines. Contrary to this, horizontal scaling is about adding more machines in order to increase processing power. A limitation to vertical scaling is that it requires expensive components and will eventually stop scaling when the latest upgrades have been added. Horizontal scaling, on the other hand, will keep on scaling just by adding commodity hardware, more computers, in theory forever. Horizontal scaling is obviously more desirable but requires the processing to be done in parallel on each machine. Writing parallel programs is tricky and distributing the processing to a cluster of machines is outright hard. Luckily, open-source systems exist which take care of the problem and expose an easy interface to the user for writing programs. Even the cluster itself, the machines, can be created easily using third-party systems.

3.2 Apache Hadoop

Hadoop[9] is a widely used collection of tools that are used for common tasks related to scalable computing. Looking at the modules included in Hadoop gives a good indication of the challenges with scaling. Two of the modules are The Hadoop Distributed File System and Hadoop MapReduce.

3.2.1 Hadoop Distributed File System

A distributed file system enables the user to distribute files to several systems. In the case of HDFS, normal *nix file commands can be used to handle files, such as *rm* and *mv*. The files are automatically synced throughout the distribution.

3.2.2 Hadoop MapReduce

In 2004, Jeffrey Dean et al. released a white paper[10] about a programming model called MapReduce. MapReduce enforces a certain type of abstraction when computing in order to distribute the work load. The model divides the computation into two steps, map and reduce. These two simple functions can represent most computations when used in combinations. Hadoop includes its own version of the popular programming model named Hadoop MapReduce.

3.3 Apache Spark

Apache Spark was open sourced in 2010 and has grown into a fierce competitor to current frameworks. Spark works well with the usual Hadoop modules but has its own processing framework. Spark's processing framework focuses on information flow[11] instead of MapReduce. The information flow often results in increased speed and a more natural way of reasoning about computing. It provides the developer with an easy interface

accessible through Scala, Java, and Python and has a complete machine learning library built-in.

3.4 Clustering

Most distributed frameworks, such as Hadoop and Spark, use the concept of clusters. A cluster is a group of connected entities which perform a task together. In the case of Hadoop and Spark, the cluster is a collection of computing nodes (computers) which distributes the workload. Both frameworks enable an easy way of creating clusters and then running jobs on them. It is usually inconvenient to maintain physical computers. Third-party providers have grown from the need of easy access to computers. One of the most known and used such companies is Amazon and its service Amazon Elastic Compute Cloud (EC2). EC2 lets its users rent computers by the hour and multiple computers may be spawned and removed trivially. Spark includes a script and a document[12] which guides the users through the initial configuration and ultimately to a work-flow consisting of three commands for running jobs.

Listing 3.1: Running Spark on EC2: Launching a cluster

```
1 ./spark-ec2 -k <keypair> -i <key-file> -s <num-slaves>
     launch <cluster-name>
```

The first command is used to create the cluster. It includes argument for cluster size, type, authentication etc. The documentation explains all the steps in detail.

Listing 3.2: Running Spark on EC2: Login to a cluster

```
1 ./spark-ec2 -k <keypair> -i <key-file> login <cluster-name>
```

In order to run jobs on the cluster, the user first needs to login via SSH[13]. The user then run Spark jobs as usual via the command line when logged in.

Listing 3.3: Running Spark on EC2: Destroying a cluster

```
1 ./spark-ec2 destroy <cluster-name>
```

Destroying the cluster, or pausing it, is a good idea since Amazon bills by used hour. Both Spark and Amazon offer great documentation on how to use its services via their respective websites[1][14].

3.5 Fault tolerance

The risk of failure increases when the complexity of a problem grows. When dealing with a large cluster, things are bound to break eventually. When things break, it leads to errors, or worse, faulty results. It is important then to pick a system which has a carefully considered fault strategy. Both MapReduce and Spark offer fault tolerance on multiple levels with precautions such as restarting nodes and exiting the whole system. Exiting the whole system might sound undesirable but is actually more helpful then silent errors which in worst case corrupt the result.

4

Machine learning

The techniques and aims of machine learning are much the same as those in statistics and applied mathematics. Essentially, machine learning concerns drawing statistical conclusions about data, also known as statistical inference. This means that there is no universal approach to machine learning but rather a set of tools. It also means that one tool may solve various different problems. Three common tools for solving problems in this manner are classification, regression analysis, and recommender systems.

First, some general topics in the area will be discussed (Section 4.1). After that the three major techniques picked for this report will be demonstrated starting with classification (Section 4.2) followed by regression analysis (Section 4.3) and recommender systems (Section 4.4).

4.1 General topics

Some features and techniques are common for most machine learning algorithms. General topics will explore some of the different categories and properties of machine learning algorithms. The subsection will also describe a few important concepts shared between all methods.

4.1.1 Feature vector

In order to quantify and represent the dataset as numbers, which can be used by the algorithms, one translates the relevant data into a vector of numbers. Each number in the vector is called a feature, and as the name hints at, it will decide the result. The technique of translating the dataset to feature vectors varies and depends on the dataset. For example, it is common to use the bag-of-words model[15] when dealing with text.

The SMHI dataset has a natural feature vector representation with each column used as a feature.

4.1.2 Supervised- and unsupervised learning

Machine learning algorithms can be divided into different categories depending on how they work and what they achieve. Two categories often used are supervised- and unsupervised algorithms. Supervised algorithms are discrete. Data is given a class and prediction data generates one of the used classes. Unsupervised algorithms are used when the goal is to understand the data itself, for example what groupings exist within the data.

4.1.3 Parametric- and non-parametric algorithms

Parametric algorithms are bound to the parameters set by the user. For example a linear regression classification algorithm will yield a binary result. A non-parametric algorithm's model will grow with its dataset. Non-parametric algorithms include nearest neighbor classifiers and random forests. In general, non-parametric algorithms are more accurate but slower.

4.1.4 Similarity measurements

A common task of machine learning algorithms is to compare the similarity of different feature vectors. While there is a few similarity functions which are used most of the time, there are no single perfect one and each case requires some consideration regarding the similarity function. Spertus et al. [4] performed a study on similarity functions which showed very good results for the cosine similarity function in particular.

4.1.5 Algorithm complexity

An important property to consider with all algorithms is its resource complexity, such as space- and time complexity. It indicates how resource usage grows with increased input. The complexity varies between algorithms and they all have different trade-offs, for example worse time complexity but higher accuracy. It is common to speak of two different speeds when looking at an algorithm in machine learning. First, its train speed. This is the time it takes for the algorithm to be trained given input. Second, the prediction speed is important. The predict speed is the time it takes to make a prediction given a trained model. It is important to have these complexities in mind as they dictate the suitability for different algorithms expected to achieve certain performance.

4.2 Classification

The idea with classification is to connect new observations to classes with the help of previous training data. The classification algorithm is often called a classifier. The features, or properties, of the data can be expressed in different forms, for example A, B,

AB or O for blood types or as integer values like the SMHI dataset. The case-study uses a binary classifier, that is, a classifier which has two possible labels: storm or not storm.

It is possible to perform forecasting using classification. The technique used in the case-study classified weather data as pre-storm data or non-pre-storm data which could be used to compute a probability to forecast a storm based on new observations. Classification does not have to be binary as there could be several possible categories for the classifier to decide where new observations should be placed. Figure 4.1 illustrates how a binary classification places mail in a spam folder instead of the users' inbox.

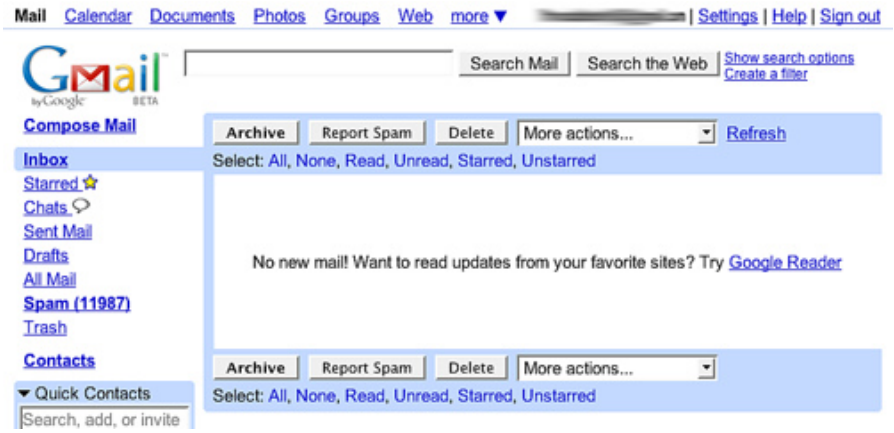

Figure 4.1: Spam filter

4.2.1 Binary- and multi-class classification

Binary- and multi-class classification are both concerned with placing observations in a correct class. Binary classification consists of two classes while multi-class classification consist of two or more classes. For example, doctors perform a multi-class classification assigning medical diagnosis given data from their previous patients. An extension to the spam example above is if the mail is classified into more classes, such as family, private, work, and spam. Some algorithms are used for binary classification while others are used for multi-class classification. It is even possible to use multiple binary classification algorithms in order to do multi-class classification.

4.2.2 Probabilistic classification

Algorithms that use probabilistic classification will return not just the class, but also the probability of that class. The probability value can be used in various ways to improve the results of the algorithms and provide feedback to improve the classifier, for instance to lower error prorogation (uncertainty) which can be avoided if the probability value is low and no conclusions can be drawn form the results.[16]

4.2.3 Algorithms

There are a lot of techniques to be used for classification in a wide range of complexity. There are linear classifications similar to the ones used for regression analysis and k-nearest neighbors which is also used for recommender systems that can be applied to classification problems. Our case-study focuses on decision trees and a combination of decision trees into Random Forests[17]. Random Forest is one of the most powerful methods for classification and can be very efficient with default parameter settings.

4.3 Regression analysis

Today, more than ever, organizations are interested in trying to predict the future. A few well known cases where prediction is used are stock prices, future sales, and climate change. The common denominator of these problems is the ability to describe them as functions. Therefore, when trying to solve or study, one of these problems what we are actually trying to do is to reproduce the function that describes the input data. Regression analysis contains several techniques for calculating the function, or curve, to fit the input data. In this section we will show some basic examples for predicting the temperature for a city based on the historical data provided. The figure below is an example of regression analysis used to predict trends in the stock market.

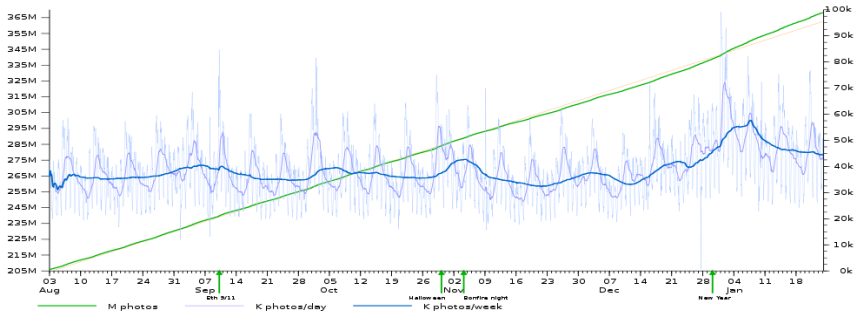

Figure 4.2: Looking for trends in time-series with the help of linear regression analysis

4.3.1 Curves and techniques

This report will focus on the *Ordinary least squares* technique to compute the constants for each x-factor in the model.[18] The ordinary least squares method is well suited since it is well documented and quite easy to grasp. Combined with the method's wide use makes it a great fit for this report.

Similar to the mathematical definition, we split curves into two categories: Linear and Non-linear curves. Linear curves are all curves on the form $y = \boldsymbol{a} * \boldsymbol{X}$ where the length of vector $\boldsymbol{a}$ and $\boldsymbol{X}$ is the same as the largest x-factor exponent in the curve. Non-linear curves are curves on the form $y = f(x)$. Examples of these functions are sine,

cosine, logarithmic functions or any other function that depends on an input variable.

Choosing the type of curve to model the data after is a critical point in regression analysis. A helpful way to decide a good curve type is to visualize the dataset. Try to see what type of curve may be a good candidate. It is of course possible to check the result for many different type of curves. The problem is that there does not exist a real end to how complex the function describing the curve can be. The complexity-versus-accuracy problem will be discussed later on but a rule of thumb is to not mix different types of functions if possible. In this report the focus will be towards polynomial curves since they have a wide area of use in practice. The difference in implementation between the curves is fairly small.

4.3.2 Core model

This section will describe how the computations of the ordinary least squares method are done to give an understanding of what is happening "under the hood". The information needed from the dataset is a feature-vector. The feature-vector contains the values we want to use to compute the curve. In the use-case we extracted the average temperature of April for each year from a city. The average temperatures will be the feature-vector in the use-case. Another important thing to be aware of is the design-vector. The design-vector contains the values for where along the x-axis each data-point in the feature-vector should be plotted. In the use-case this came straightforward from the feature-vector. We took the first data-point as index 0 and then each consecutive point got the corresponding natural number. However there are two pitfalls to be aware of. The first comes back to chapter two about missing values. In the use-case, if a year would not have an average temperature for April the design-vector would be wrongly adjusted one step. This would possibly end up in an incorrect curve. The other thing is if the design-vector does not follow a uniform tick between each data-point. This can usually be determined from the context of the dataset.

We will make up a simple example to explain the computations of the ordinary least squares method and to show how the underlying vectors look. The vectors will be a feature-vector with five values and a design-vector with five consecutive values for simplicity. They could look as follows;

$$Feature - vector = \begin{pmatrix} 3 \\ 4 \\ 6 \\ 8 \\ 11 \end{pmatrix} Design - vector = \begin{pmatrix} 0 \\ 1 \\ 2 \\ 3 \\ 4 \end{pmatrix}$$

Now the formula for the ordinary least squares method is $(X'X)^{-1}X'Y$. Y is the

same matrix as the feature-vector, just transposed to create the correct output matrix. The X matrix is the same as the design-vector with one minor change. In the X matrix we determine the exponent factor of x in the computed model. So if we want to make a classic linear equation on the form $y = k * x + m$ or more explicitly $y = k * x^1 + m * x^0$. Then the X matrix will look as the figure below. X' is the transpose of X and $(X'X)^-1$ is the inversion of the dotproduct of X and it's inversion. The X matrix for the line we want to compute looks as follows;

$$X = \begin{array}{c} \begin{array}{cc} x^0 & x^1 \end{array} \\ \begin{pmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \\ 1 & 4 \\ 1 & 5 \end{pmatrix} \end{array}$$

Now that we know how the matrices look for each variable in the formula the actual calculations in itself will not be shown here. Rather we will show a few lines of code that perform the calculations.[19]

```
1 yMat = mat(valueArr).T
2 xTx = xMat.T*xMat
3 ws = xTx.I * (xMat.T*yMat)
```

ws is an $1 \times n$ matrix with each slope coefficient in consecutive order starting with x^0

4.3.3 Managing the model

When the system is implemented it is time to determine which curve describes the data points best. A common method to do this in regression analysis is the *coefficient of determination.*[20] The coefficient of determination is calculated as an average error-rate between the predicted points and the data-points provided. The coefficient is a value between 0 and 1 which is called R squared, written R^2. The closer R^2 is to 1, the better the curve describes the data-points. If $R^2 = 1$ then all the data-points are on the computed curve. The mathematical computation of R squared is as follows;

$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}} \quad SS_{tot} = \sum (y_i - \overline{y})^2 \quad SS_{res} = \sum (y_i - f_i)$$

R^2 is a way to check how well the model fits the training data. This can be a nice measurement to have but it is more interesting to see how well the model fits validation data. this report uses the metrics of *mean absolute error* to compute the fit of validation data (test data). The equation below shows how the mean absolute error is computed. y_i is

the data-point from the validation set and f_i is the model computed value for that index.

$$\sum \sqrt{(y_i - f_i)^2}$$

To cross-validate the models we split up the dataset into k equally sized *folds* and use $k-1$ folds for training and the last fold for validation.[21] For each run we compute the mean absolute error. The computations run k times, so each fold get to act as validation. Then we compute the average mean absolute error to get a value that is not dependent on a specific fold.

It is common to get more data over time from problems solved by regression analysis. Therefore, it is interesting to know how to update the model and get a better prediction over time. The solution is simple. Add the new data-points to the existing dataset and run the system again to get the new model. Usually this is not done for every new data-point that gets observed; it is rather done in batches. How large a batch is depends on the data intensity of the observations. Since the latest model is saved away it can still be used while computing the new model. This allows for a nice overlap when updating the model.

Saving away the model is simple. After determining which model fits the data best. The only thing needed to be done is to save it or send it to the system that will benefit from the predicted data.

4.3.4 Complexity vs accuracy trade-off

The main aim of regression analysis is not to predict the exact data-point, even if that would of course be nice, but rather to predict trends and find the way the data develops. Due to this, reason there exists a model complexity versus prediction accuracy trade-off. A complex model correlates well with *historical data*, but usually leads to overfitting and poor accuracy when trying to predict new data. This is the core problem in regression analysis. A very complex model can fit any cluster of data-points but it does not tell how well the model predicts the future.

There is no simple correct way to handle this trade-off but there are two common ways to look at it. The first one is to strive toward choosing a simple model. It is useful to use the R^2 measurement to see how much better a more complex model fits the data than a simple model. A useful guideline for linear models is to only use models that contain the first three x exponents, it is preferred that those slope coefficients may not be zero. The other tool to handle complexity and accuracy is to actually look at the curve. Measurements can only help with historical data but we can with our eyes and mind make assumptions on where data-points might come in the future. All these tools

are subject to us developers to some degree which might not be perfect, so use the tools with some caution.

4.4 Recommender system

In certain situations, data is available but with missing values. For example a user who has rated some movies, but not all. Note that these missing values are intentional and not a dataset flaw. A recommender system tries to fill in the missing values using current knowledge. It recommends new values. It is easy to reason about the problem by dividing it into a user part and an item part. The user is the user in the case of rating movies, while the movie is the item. Most datasets where recommendation systems are appropriate can be modeled this way. Two approaches which try to fill in the blanks are neighborhood-based recommendations and model-based recommendations.

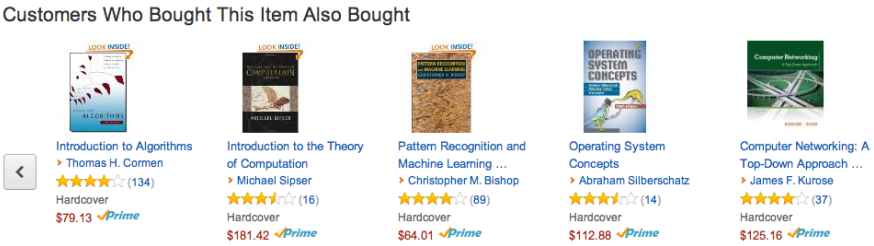

Figure 4.3: Amazon recommended items

A common use-case for recommender systems is finding items that a user might be interested in, or a movie the user probably will like with regards to previously rated movies. A wide range of services benefit from recommender systems with some even using it as its main product.

4.4.1 Neighborhood-based

This approach concerns itself with finding users that should behave similarly to the investigated user - a neighborhood. The neighbors then decide what the user's missing value should be. A common technique when looking for neighbors is to use domain-specific knowledge about the problem. For example, a social networking site might want to pick neighbors based on the friend relation with the user. A possible way of finding neighbors for our SMHI problem is to pick them based on physical distance from the user. The classification algorithm k-nearest neighbors[22] can pick neighbors for us without requiring specific domain knowledge. The algorithm returns k users based on how similar their feature vectors are, compared to the investigated user's feature vector using a similarity function described earlier.

Listing 4.1: k-nearest neighbors on Spark

```
# rdd_vectors is a rdd without the investigated vector
sims = rdd_vectors.map(lambda ary: (similarity(vector[1:],
    ary[1:]), ary[0]))
sims = sims.sortByKey(ascending=False)
most_similar = sims.take(10)
```

When the neighbors have been decided we can let them vote on what value the user should have. The kNN algorithm also returns how similar the neighbor is to the user and we can use that similarity to weigh the vote of the neighbor, similar neighbor's votes influences the result more. When using domain specific knowledge all neighbors may have the same influence, or in our examples, the number of mutual friends and physical distance.

Listing 4.2: Weighing neighbors

```
# new base to vote
base = 1 / reduce(lambda x,y: x+y[0], most_similar, 0)

# find the averages from the knns
neighborhood_averages = averages.filter(lambda ary: ary[0]
    in [y for x,y in most_similar])
closest_cities_temp_mean = neighborhood_averages.map(lambda
    ary: [[x for x,y in most_similar if (y == ary[0])][0],
    ary[39]])

# Now we weigh the votes with regards to distance
new_city_temp_mean = reduce(lambda x,y: x+(base*y[0]*y[1]),
    closest_cities_temp_mean.collect(), 0)
```

Neighborhood-based recommender systems are usually easy to understand and implement but suffer from a few practical flaws when used with big data. The whole dataset must be considered in order to find neighbors each time a prediction is made, therefor, the dataset must be **in-memory**. Neighborhood-based recommender systems are widely used even with the in-memory performance characteristic.

4.4.2 Model-based

It is common to create models for problems in math. A model usually requires some limitations and assumptions in order to be practical. When the model has been created it can be used to explore scenarios and problems related to it. A common approach when

building recommender systems is to create a model for the dataset and then use that model to, for example, predict missing values. Some models can be saved and used later.

Matrix factorization[23] creates approximate product matrices of an input matrix. The goal is for the product matrices to be equal to the input matrix when multiplied with each other.

$$R \approx P \times Q^t = \hat{R}$$

The matrix $\hat{R}$ contains the missing values from R so that recommendations can be made. Spark implements an algorithm for matrix factorization using alternating least squares[24]. The input used when training should be on the form $[user, item, rating]$.

Listing 4.3: Using Spark's matrix factorization

```
1 from pyspark.mllib.recommendation import ALS
2 # train = [[user, item, rating], [user, item, rating] ...]
3 model = ALS.train(train, 2, 25)
```

The second argument passed to the method is the number of desired latent factors[25] while the third argument indicates how many times the approximation algorithm will run. As always, it is important to test and tune these arguments to find a good balance between accuracy and performance.

The prediction only requires the user and item as expected.

Listing 4.4: Predicting with Spark's matrix factorization

```
1 model.predict(user, item)
```

4.4.3 Explicit- and implicit data

Explicit- and implicit data collection are the two ways of gathering data for a recommender system. Explicit ratings are the direct data in relation to an item, such as a vote, while implicit data is gathered by assuming causation of user actions. When a user for example watches a lot of romantic movies and that is thought to affect the user's preference, then that is implicit recommender data. It is easier to gather implicit data than explicit data, but implicit data is usually less accurate and using it will sometimes infringe on the user's privacy.

5

Case-study results

5.1 Evaluation strategy

Classification, regression analysis, and recommender system each use their own described technique in order to evalute accuracy and result. The most common method however is to use cross-validation[26]. One of the most common ways to deal with this is to split up the dataset into *k-folds*. $k-1$ folds is used for training the algorithm and the last fold is used for validating how the algorithm preforms. To get an average result of how good the algorithms perform the validating fold is switched so each fold gets to validate one time each and then the average is computed as a result.

5.2 Classification

A decision was made to go for the Random Forest[17] algorithm since it has become one of the most popular algorithms lately. Random forest performs well with noise and variable scaling so depending on how we set up our test we would still be able to use the same algorithm if we should change our minds and try a different implementation. Random Forest combine decision trees into a forest. Generated forests can be saved for future use and estimates of what variables are important, are some of the Features of Random Forests[**?**].

There is no implementation of Random Forest in Sparks Mllib[27], thrilled by the idea to implement our own Random Forest algorithm for spark, we decided not to, due time limitations. This means that following tests are done by scikit-learn[28], pandas[29] and NumPy[30] in python.

The first attempt did not show any sign of any storms in Sweden for the specific test city. Problem were that there the algorithms could not find 5 storms in a total of

137,465 data values even though adjustments were made and the algorithms was left to run for several minutes. According to SMHI the criteria for a storm is when wind force exceed 24,5 m/s (10-11 Beaufort)[31], this is translated into >24 since there is no double values for wind force in our datasets.

To achieve better results in the second attempt there were changes to be made. The dataset was adjusted so that there were significant more storm values in the training set, which would create a better classifier to be used for test set. A similar technique is used in Williams, John K et al[32], where test data is created in addition to existing data to get a more clever algorithm and better end results.

The second attempt gave much better results than the first attempt where the classification looked like the table 5.1 below. In table 5.2 there are two instances that have been classified as storm and there is none in first try. Note: both tables are fragments of the two generated in the test (>130,000 rows).

Table 5.1: First attempt

Storm	Non-storm
1.000	0.000
1.000	0.000
9.999	0.001
1.000	0.000
9.998	0.002

Table 5.2: Second attempt

Storm	Non-storm
1.000	0.000
9.974	0.026
0.005	9.995
0.007	9.993
9.989	0.011

A binary count using NumPy were done to see how many storm that were not found in the first attempt and how many was were found in second attempt.

The main goal was to see if we could predict storms from given weather data, applied to another test weather station (unseen data for our algorithm) to see if we could predict any storms for that location. Results show that we found more storms than there actually were during time of data. In a closer look at the dataset we found in total 3 storms, 14 points where wind exceeded 20 m/s and 3216 where it exceeded 15m/s.

Falsterbo, one of the weather stations with a total of 20 measurements indicating storm had more than four times of predictions which is close to the number of measured points where wind force exceed 20m/s (171).

5.3 Regression analysis

An abstraction is made over the dataset by using average temperatures within different timespans. First, by years. Then by using average temperature. The final set of data points is an average of the month of April each year. This became another abstraction layer in the sense of taking samples, which was necessary. The best model was determined with the combined result of the R^2 value, the value of *the mean absolute error* and subjective prediction of the curve.

The X^4-model would be the best fit only looking at the R^2 value. When taking into account the mean absolute error from the cross-validation, the X^2-model performs better at prediction. We do note the steep upward trend that starts in the end, but it is clearly less steep than the one computed by the X^4-model. So among these models, the X^2-model is the preferred one.

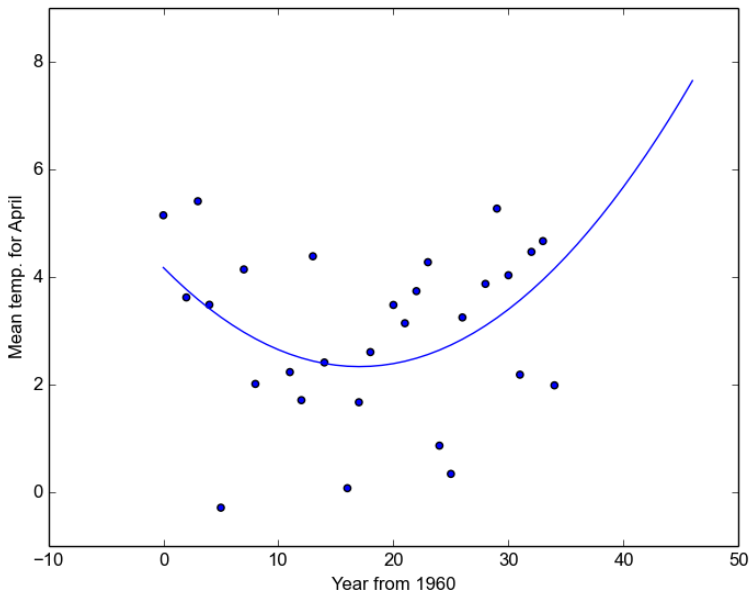

Figure 5.1: X^2-model with $R^2 = 0.1344$ and Mean absolute error = 0.9469

5.4 Recommender system

Each method is trained on 80% of the data and tested on the remaining 20%. All data and categories are picked at random.

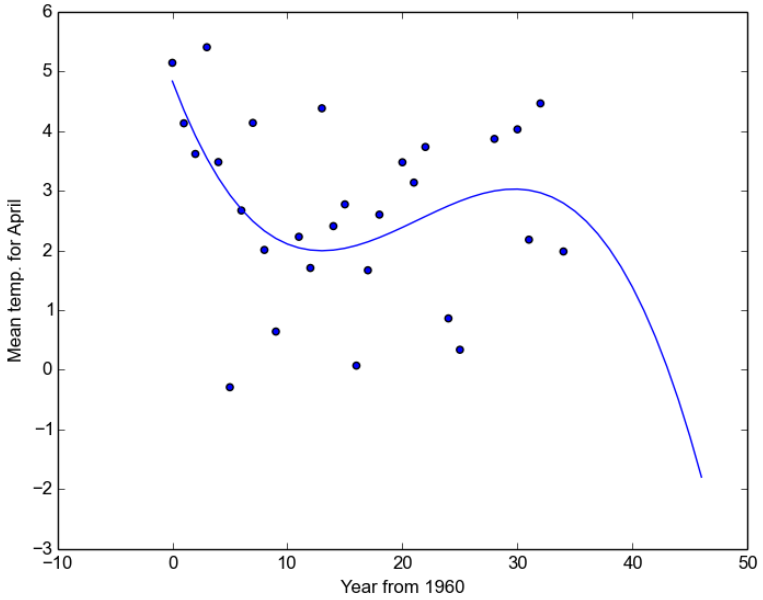

Figure 5.2: X^3-model with $R^2 = 0.2243$ and Mean absolute error = 1.287

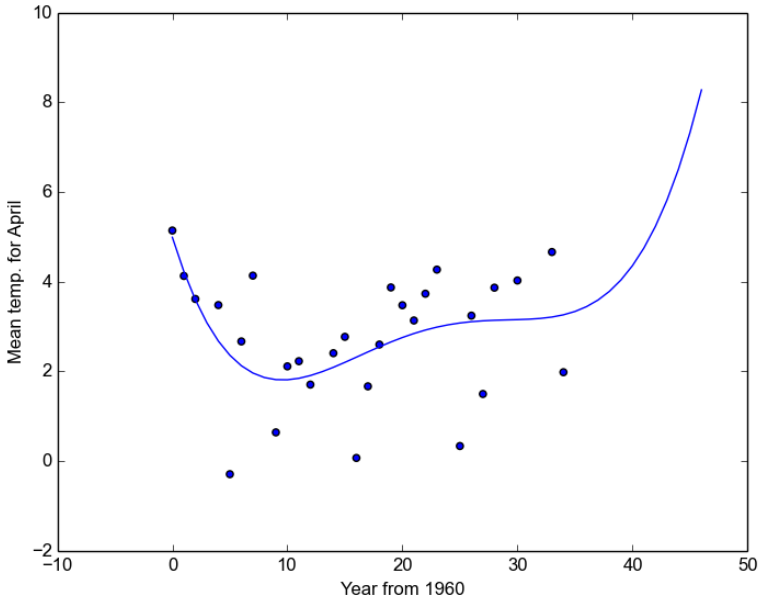

Figure 5.3: X^4-model with $R^2 = 0.2452$ and Mean absolute error = 1.089

5.4.1 Neighborhood-based

First n neighbors are found using the k-nearest neighbor algorithm with features other than the one being predicted. The neighbors do a weighted vote on the missing feature. The weight is determined based on the similarity of the k-nearest neighbor algorithm. All cities are tested in turn against the others in each test.

The lowest error was achieved when 5 neighbors were picked with the kNN algorithm. If a new city is inserted into the dataset without a temperature, then we can approximately predict its actual temperature almost within one degree Celsius. The graph 5.1 shows a slight increase in error as the number of kNN neighbors increases. The increase in error will probably continue to grow as we add more neighbors.

kNN neighbors	error
3	1.1
5	1.09
10	1.21
15	1.34

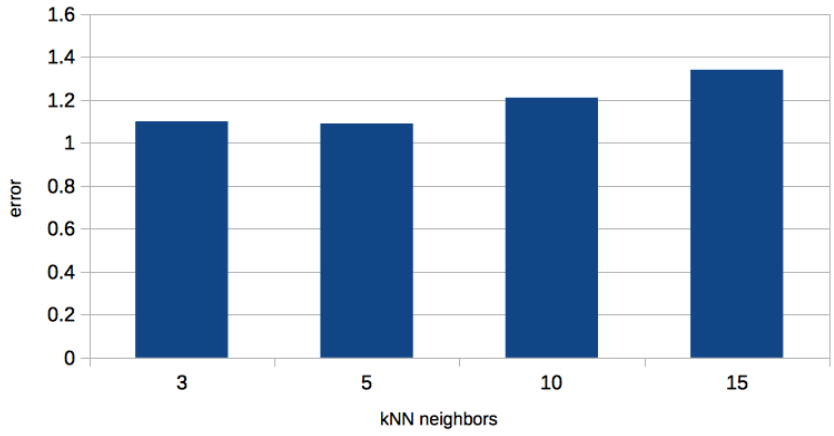

Figure 5.4: kNN neighbors and error

5.4.2 Model-based

Each city inputs each of its features as a ranking into the matrix factorization algorithm. The investigated city's feature is not added. Then the city's missing feature is predicted

with the help of the newly constructed matrix.

Using 40 latent factors produced an average error of 0.268 looking at the temperature feature (item). If a new city is inserted into the dataset without a temperature, then we can approximately predict its actual temperature almost within a quarter of a degree Celsius. In the graph Figure 5.2, we can clearly see that using between 15 and 43 latent factors seems to produce good results.

latent factors	error
1	2.53
2	2.26
5	1.79
10	4.27
15	0.65
20	0.37
25	0.33
30	0.33
35	0.30
40	0.26
43	0.35
45	1.55
50	1.87

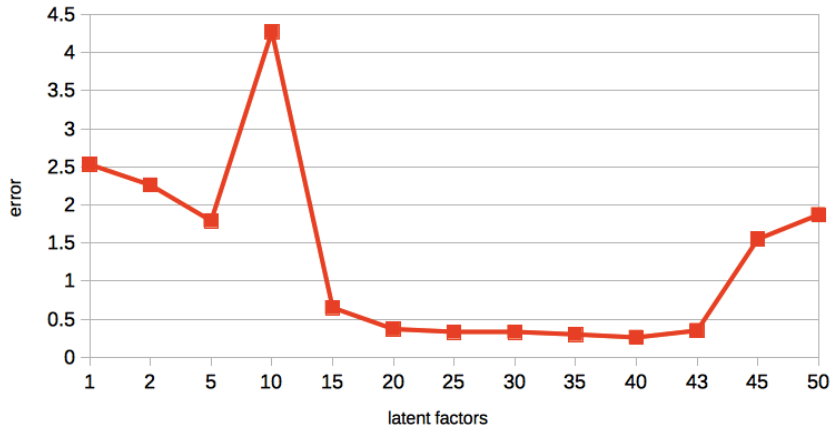

Figure 5.5: latent factors and error

6

Discussion

The goal with the report was to create a tool for organizations that want to get started with machine learning. The report divides the problem into three different steps. Each step contains relevant and up-to-date descriptions, tips, caveats, and guidelines. By the end of the report, enough content is covered in order to allow the reader to solve the original problem. This complies with the goal of the report and it should be deemed successful.

A sub-goal of the report was to provide a good case-study for demonstrative purposes. The case-study lacks some details and scientific reproducibility, but it is easy to understand and it demonstrates the discussed problems clearly. Although not optimal, this compiles with the goals for the case-study.

6.1 Target audience and the need for scale

We intended this report to be read by people that want to use their data but do not know how. Most chapters of the book are really useful for such people. The chapter about scalability is only useful if the reader has more data than what can be stored in memory, or even on a hard drive. In practice, organizations with this amount of data usually have sufficient knowledge and resources to make this report redundant. We believe that the target audience may still have use for the report as scalable machine learning is a very specialized topic across disciplines. The assumption of sufficient resources at organizations may become flawed as data gathering gets increasingly accessible to more people.

6.2 Scientific-, social- and ethical aspects

This report has kept a clear focus of the benefits that comes with machine learning and big data. But storing data is a hot subject in media for all the wrong reasons. The following thoughts and discussion reflect the potential downsides in using and storing user data.

We look at machine learning and big data with eyes filled with excitement and eager, as an opportunity to create and improve. Unfortunately, technology is not restricted to good doings. Without proper care, human rights can be jeopardized with or without evil intent.

Hollywood brought us Skynet and the discussion about malicious robots revolting against humans. Even though we are nowhere near strong artificial intelligence, modern computers and techniques, including those described in this paper, enables unimaginable storage capacity. There is no longer a need for ever throwing away a piece of information gathered about a user. As described in this paper, increasing data quantity paired with the rights tools makes for really good features - and weapons. Laws aimed at protecting the privacy of people have never been as important as they are now. Entities gathering information have never had as large responsibility as they have now.

Currently, the control lies in the hands of companies and governments and most of the rules are made with share holders and digital warfare in mind. The step from a great recommendation on an e-commerce site to unfair profiling using private information is small. Sometimes even reasonable systems can be used for questionable purposes, e.g. do you really want your insurance company to run machine learning on your genetics data?

At the time of writing, news of countries (U.S.A) doing surveillance on a global level is getting more common. Spying is no longer about knowing what your enemies are doing, but also knowing what your friends are doing and what your people are doing. The consequences of these actions are beginning to show in the topics discussed at meetings originally intended to increase the wellbeing of people. The last time international laws surrounding similar attitudes was agreed on date back to 1949 - they where called the Geneva Conventions[33].

Even though these extreme scenarios are possible we still believe that the great possibilities that come with machine learning and big data outperform the downsides. We do not think the right way to tackle this problem is by trying to put boundaries on the technology. History has shown that it is much too complicated with the current speed of progress. We rather point out the importance of informing and educating people on how to act on the Internet and to be more aware about their privacy. A quote from Eric Schmidt may sound like a cliché but is really worth considering, “If you have some-

thing that you don't want anyone to know, maybe you shouldn't be doing it in the first place"[34].

The era of machine learning and big data is in an early stage and the possibilities that come from it are still not widely discovered. We hope and think that this report will benefit organizations that still have not found a good use for their data. We think this will truly expand and push the limits of what is possible in machine learning thanks to innovation in every day.

6.3 Case-study shortcuts

A few shortcuts were taken when creating the case-study. They where needed so that certain concepts would be clearer for the reader, but also to reduce unnecessary work for the authors. It was never a goal for the report to benchmark nor to optimize the methods described. Unfortunately, this makes the case-study harder to reproduce and ultimately verify.

6.4 System

The system and operations surrounding production machine learning for big data may arguably be a harder challenge than the modeling itself, which we focus on in this report. The math is often proven and even neatly packaged, however, the infrastructure from raw data to real-time queries is not. There are few frameworks helping with this general, and vast, problem. The final solution is often unique and formed by legacy technologies and methodologies.

6.5 Streaming

Using real-time data with machine learning is called adaptive machine learning. Real-time data can also be used with visualization instead of sampling. Streaming is a complex challenge and outside the scope of this report.

6.6 Further work

Streaming is an interesting field and its perspective would add significant complexity, but also value, to different topics such as visualization, scalability, and machine learning.

The methods in this report only constitutes the tip of the iceberg. We want to add more algorithms which solves more problems for the reader. There are plenty of scalability alternatives not discussed in this report which might fit certain organizations better. Ultimately, the format of a book would be preferable to that of a report.

6.7 Conclusion

Looking back is not always easy. Knowing where one started then ended up can sometimes be harsh. But in our case it is rather joyful. Our goal was clear from the beginning: take on one of the more advanced topics in computer science that is still under heavy development, and create a document for organizations that is both guiding and fun to read. We are confident in succeeding with that goal.

We have had some initial feedback from the community and the overall attitude has been positive and encouraging. Representatives from AstraZeneca and Teradata have expressed excitement and praise. There are separate resources available for most of the topics discussed in this report. We felt a more comprehensive and complete document was needed and it ultimately tied the topics together nicely. We have not been able to find a document with similar breadth.

The report helps the reader go from data to value in logical steps. A mostly complete system can be built by following this report. Of course, there is a lot more to explore. Each topic comes with its own set of challenges and a report like this could easily be turned into several books. The goal was never to fully emerge into each topic but to give a brief overview helping the reader to explore on her own. It was important to open the eyes of the reader and disarm the different topics.

Bibliography

[1] Apache spark (Apr. 2014).
URL http://spark.apache.org/

[2] Smhi weather data (Apr. 2014).
URL http://www.smhi.se/klimatdata/Oppna-data/Meteorologiska-data

[3] Correlation does not imply causation (Apr. 2014).
URL http://en.wikipedia.org/wiki/Correlation_does_not_imply_causation

[4] C. R. Edgar Acuña, The treatment of missing values and its effect in the classifier accuracy, Tech. rep., Department of Mathematics, University of Puerto Rico at Mayaguez, Puerto Rico (2004).

[5] A. C. Acock, Working with missing values, Tech. rep., Department of Human Development and Family Sciences, Oregon State University, Corvallis (2005).

[6] Anscombe's quartet (Apr. 2014).
URL http://en.wikipedia.org/wiki/Anscombe's_quartet

[7] Support vector machine (May 2014).
URL http://en.wikipedia.org/wiki/Support_vector_machine

[8] matplotlib (Apr. 2014).
URL http://matplotlib.org/

[9] Hadoop (Apr. 2014).
URL http://hadoop.apache.org/

[10] S. G. Jeffrey Dean, Mapreduce: Simplied data processing on large clusters, Tech. rep., Google (2004).

[11] M. J. F. S. S. Matei Zaharia, Mosharaf Chowdhury, I. Stoica, Spark: Cluster computing with working sets, Tech. rep., University of California, Berkeley (2010).

[12] Running spark on ec2 (Apr. 2014).
URL `http://spark.apache.org/docs/latest/ec2-scripts.html`

[13] Secure shell (Apr. 2014).
URL `http://en.wikipedia.org/wiki/Secure_Shell`

[14] Amazon web services (Apr. 2014).
URL `https://aws.amazon.com/`

[15] Bag-of-words model (Apr. 2014).
URL `http://en.wikipedia.org/wiki/Bag-of-words_model`

[16] O. B. Ellen Spertus, Mehran Sahami, Evaluating similarity measures: A large-scale study in the orkut social network, Tech. rep., Mills College, Google (2005).

[17] A. Nicluscu-Mizil, R. Caruana, Predicting good probabilities with supervised learning, Tech. rep., Department of Computer Science, Cornell University, Itchaca NY 14853 (2005).

[18] L. Breiman, Random forests, Tech. rep., Statistics Department, University of California, Berkeley (2001).

[19] Ordinary least squares (May 2014).
URL `http://en.wikipedia.org/wiki/Ordinary_least_squares`

[20] P. Harrington, Machine Learning in action, Manning, 2012.

[21] Coefficient of determination (May 2014).
URL `http://en.wikipedia.org/wiki/Coefficient_of_determination`

[22] Cross-validation (May 2014).
URL `http://en.wikipedia.org/wiki/Cross-validation_%28statistics%29`

[23] k-nearest neighbors (Apr. 2014).
URL `http://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm/`

[24] Matrix decomposition (Apr. 2014).
URL `http://en.wikipedia.org/wiki/Matrix_decomposition`

[25] M. J. Dave Zachariah, Martin Sundin, S. Chatterjee, Alternating least-squares for low-rank matrix reconstruction, Tech. rep., ACCESS Linnaeus Centre, KTH Royal Institute of Technology (2012).

[26] Latent variable (Apr. 2014).
URL `http://en.wikipedia.org/wiki/Latent_variable`

[27] Cross-validation (Apr. 2014).
URL `http://en.wikipedia.org/wiki/Cross-validation_(statistics)`

[28] Random forests, leo breiman and adele cutler (May 2014).
URL `http://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm#workings`

[29] Spark mllib documentation 0.9 (May 2014).
URL `http://spark.apache.org/docs/0.9.0/api/mllib/index.html`

[30] scikit-learn, machine learning in python (May 2014).
URL `http://scikit-learn.org/stable/`

[31] pandas (May 2014).
URL `http://pandas.pydata.org/`

[32] Numpy (2014).
URL `http://www.numpy.org/`

[33] About storms smhi (May 2014).
URL `http://www.smhi.se/kunskapsbanken/vad-betyder-olika-ord-om-vind-1.35876`

[34] J. K. Williams, T. S. M. S. D.A Ahijevych, C. J. Kessinger, S. Detting, A machine learning approach to finding weather regimes and skillful predictor combinations for short-term storm forecasting, Tech. rep., National Center for Atmospheric Research, Boulder, Colorado (2008).

[35] The geneva conventions of 1949 and their additional protocols (May 2014).
URL `http://www.icrc.org/eng/war-and-law/treaties-customary-law/geneva-conventions/`

[36] Eric schmidt about privacy online (May 2014).
URL `http://video.cnbc.com/gallery/?video=1372176413`