UNIVERSITY OF SOUTHERN DENMARK

SOFTWARE ENGINEERING 6. SEMESTER

Datamining and its use

Author:
Lasse Bjørn HANSEN
Simon FLENSTED

Supervisor:
Jan Corfixen Sørensen SØRENSEN

UNIVERSITY OF SOUTHERN DENMARK

A report submitted in fulfillment of the requirements
of Software Engineering 6. semester

at

University of Southern Denmark
TEK

May 10, 2017

"Some quote"

- Gruppe 3

Contents

Chapter 1

Introduction

1.1 Motivation

The amount of data being processed around the Internet and within big systems is continuously increasing. This data should be structured and modelled in a way that makes it easily accessible and easy to work with. Handling large amounts of data the right way can prove to be very useful, not only to the company who possess the data, but also to the end users of a product. To achieve this, the art of data mining is very useful. The company Struct A/S have provided a typical software engineering task where data mining will create the foundation. This report will address theoretical aspects about data mining, how it is done in practice and how the final results of the processed data can be put to use. [1]

1.1.1 Data mining

Data mining has become a big part of modern software engineering. Lots of companies tends to store large amount of data without structure and order within the data. This results in a lot of useless data which is both ineffective and a waste of resources. With prober data mining, it is possible to make this useless data useful to the company and its end users. In this case the data contain valuable information about users visiting the websites created and hosted by Struct A/S. By processing the data properly, it can be used for product recommendation, among other things.(Find kilde på dette)

1.1.2 Product recommendation

If an e-commerce company wants to increase its profit, there is no doubt that product recommendation is one of the better ways of increasing your profit. This was first made popular by the retail giant Amazon. If you can predict what sorts of products your costumer may find useful, additional sales becomes more frequent. Big data sets, like the one provided by Struct A/S, can make it possible to predict customer needs, if the data is processed properly.(Find kilde på dette)

Chapter 2

Problem statement

2.1 Problem description

The initial problem/challenge is given to us by the company Struct A/S and is described as follows:

When launching sites, whether it being regular websites or web shops, a lot of user activity is logged. We therefore have a large amount of data associated with each of our sites but do not currently use it.
In the future we would like to be able to use logged data to generate an insight into the user activity on our site and actively use this data to create a personalized experience for the users.

This project will handle the initial normalization of the data, storing it in a scalable way and utilizing the data to create features which add value for the company and the users. In order to achieve this, theory has to become implementation. Research is required in terms of data storing, data mining and recommendation algorithms. This research is implemented in the end system creating an API allowing Struct A/S to get useful information from the data such as the recommended products for a certain user. This API will be the final product and will utilize different technologies and algorithms.

2.2 Problem statement

The data we have been given is in a de-normalized format and the problem therefore comes with two challenges - normalizing the data and utilizing the data to create a personalized experience for the users.
This leads to the following research questions:

- How can you effectively normalize large amounts of data?
- How can you optimally store and access data in a scalable way?
- How can you utilize the data to generate useful features for the company and the end user?

Chapter 3

Implementation

3.1 Solution overview

The final solution consists of a *RESTful* API build with ASP.NET Core and a MongoDB *No-SQL* database. Both are hosted through Amazon Webservices in an EC2 container using Docker.
The API consists of the commands seen in appendix A.

This section covers the process from the initial data delivery to the final solution in a chronological order.

3.2 Initial data dump

In the beginning of the project we were supplied with data from one of *Struct A/S* customers. This data was in the form of many SQL tables and most of the data was not relevant for creating product recommendations. The main tables used were the following:

- Visitor: A collection of every unique visitor who visited the customer's website, each visitor gets a unique identifier called UID. Contains 3,073,665 visitors.
- Profile: A collection of every users signed up at the website. Contains 3037 profiles.
- BehaviorData: A collection of every unique action performed by visitors on the website, for example when a visitor views a product a new row is made with the visitor's UID, the product UID and the timestamp. Contains 3,326,736 visitor actions.
- Order: Contains each profile's orders. Contains 5520 orders.
- Product: A collection of all products on the website with their unique IDs. Contains 22,445 products.
- ProductGroup: A collection of all product groups. Contains 262 product groups.
- AttributeValueRendered: A collection of product and product group descriptions in different languages. Contains 409,259 descriptions:

A snapshot of the visitor and behaviorData table can be seen in figure 3.1 and 3.2.

	Uid	UserAgent	BrowserName	BrowserVersion	IPaddress
1	6820EDD0-E6A6-4105-A078-0000127E7AE1	AdsBot-Google (+http://www.google.com/adsbot.html)	Unknown	0.0	66.249.[redacted]
2	4D9FC023-9034-4B93-96D8-000013AB1940	Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.3...	Chrome	36.0	212.71.[redacted]
3	CC43266B-5E13-473E-A4F2-000019D2B065	Mozilla/5.0 (compatible; Googlebot/2.1; +http://ww...	Mozilla	0.0	66.249.[redacted]

FIGURE 3.1: Visitor table from the original data

	Type	Id	UserId	Timestamp	ActivityData
1	ProductView	31070	794ebe68-49c5-451c-ba80-226cdc0508f4	2016-09-30 20:04:01.5376346 +02:00	NULL
2	ProductView	31071	055bf3eb-a556-4672-9f65-5f64932e3973	2017-01-05 12:18:05.5065939 +01:00	NULL
3	ProductView	31071	06d3f1b4-8347-499d-9edd-71c058024cc1	2017-01-27 10:42:00.8031017 +01:00	NULL

FIGURE 3.2: Behavior data table from the original data

These tables contain all the pertinent information for creating product recommendations and can be utilized after a cleaning and structuring process. This process is described in the following sections.

3.3 Data Transformation

To begin the initial data transformation a data storing technology has to be selected. The technology chosen was *No-SQL*, specifically *MongoDB* the most popular No-SQL framework [4].
No-SQL is chosen because of the good fit for this project. The data demands are not clearly specified in the beginning and with No-SQL it is easy to add or remove data or even change the data types on the fly. No-SQL's denormalized format also allows for faster retrieval of a single item without having to do joins or complex SQL queries. Finally No-SQL is easier to scale across multiple servers and many engines have built in scaling functionalities [3] which can come in handy when multiple clients begin using the service.

A brief overview of the different terminology for SQL and No-SQL is given in table 3.1.

TABLE 3.1: SQL vs No-SQL terminology

SQL	**No-SQL**	**Comment**
Table	Collection	
Row	Document	A No-SQL document can contain more complex datatype compared to a row in SQL e.g arrays or other documents

Python was used to accomplish the early migration from SQL tables to MongoDB. Several scripts were created to retrieve the data from the SQL server and transfer it to the MongoDB database in the wanted format. Pseudo code of one of these scripts can be seen in algorithm 1.

After all the scripts are finished the No-SQL database has the collections Visitor and Product. A breakdown of documents in the two collections can be seen in table 3.2

Algorithm 1 Product Script

```
SQLquery = SELECT * FROM struct.Product
db = MongoDB
for all Rows r in SQLquery do
    Product = {Id: r.id
            Description: ""
            Created: r.created
            visitorID: []
            ProductGroupId: 0 }
    db.insert(Product)
end for
Description = ""
firstId = true
previousId = 0
previousGroupId = 0
SQLquery = SELECT * FROM struct.product JOIN struct.attributeValueRendered ON id ORDER
BY productId desc
for all Rows r in SQLquery do
    currentId = r.ProductId
    currentGroupId = r.GroupId
    if firstId then
        previousId = currentId
        firstId = false
    end if
    if currentId != previousId then
        db.update({id: previousId},
                $set: db.description: description
        db.update({id: previousId},
                $set: db.ProductGroupId: previousGroupId
        description = ""
    end if
    description += r.description
    previousId = currentId
    previousgroupId = currentGroupId
end for
VisitorIds = []
firstId = true
previousId = 0
SQLquery = SELECT * FROM struct.BehaviorData
for all Rows r in SQLquery do
    currentId = r.Id
    if firstId then
        previousId = currentId
        firstId = false
    end if
    if currentId != previousId then
        db.update({id: previousId},
                $set: db.VisitorId: visitorIds
        db.update({id: previousId},
        visitorIds= []
    end if
    visitorIds.append(r.UserId)
    previousId = currentId
end for
```

TABLE 3.2: An overview of the fields in each document in the collections

Document	Fields	Comment
Visitor	Id: string Behaviors: array ProfileUID: string CustomerUID: string	The behavior array is an array of documents with the fields Type, Id and Timestamp. This contains all the behaviors of the specific visitor
Product	Id: int ProductGroupId: int VisitorId: stringArray Description: string Created: DateTime	The visitorId array contains Ids of all visitors who have looked at this product

After the data has been cleaned and structured in No-SQL the algorithm for determining product recommendations can be made. The algorithm is described in the following section.

3.4 The product recommendation algorithm

There are multiple ways to implement af product recommendation algorithm all with their advantages and disadvantages. The method chosen for this project is called Item-to-Item collaborative filtering. Other methods and the reasoning why these weren't chosen is described in further detail in Chapter 6 discussion.

Item-to-Item collaborative filtering is a datamining tool to link items (products) with other items in terms of their similarity. This method is also the way *Amazon* handles their product recommendations [5].
The specifics of the algorithm differs from implementation to implementation. In this version each product is compared to other products based on how much they have been viewed together by customers, the likeness of their description and their product group.
The first run of the algorithm requires going through each product and the visitors of each product to see what else they have looked at. This needs a lot of resources, but once run only new behavior has to be re-calculated. A run down of the algorithm can be seen in algorithm 2.

Algorithm 2 Item-to-Item collaborative filtering algorithm

```
for all Products p do
    productScores = Dictionary<int, double>
    for all Visitors in p do
        for all Products visitorProduct in Visitor behaviors do
            if productScores contains visitorProduct then
                productScores[visitorProduct]++
            else
                productScores.Add(visitorProduct, 1)
            end if
        end for
    end for
    Sort productScores after highest value
    for all Products similarProduct in productScores do
        productScores[similarProduct] = calculateSimilarityScore(p, similarProduct, productScores[similarProduct]) (see algorithm 3)
    end for
    Sort productScores after highest value
    Store top 10 productScores in database under p
end for
```

Algorithm 3 Similarity calculations for two products

```
calculateSimilarityScore(mainProduct p1, compareProduct p2, currentScore)
similarAttributeFactor = 0.02
productGroupFactor = 0
numOfSimAttributes = 0
if p1.productGroup equals p2.productGroup then
    productGroupFactor = 0.02
end if
for all words w in p1.description do
    if w is in p2.description then
        numOfSimAttributes++
    end if
end for
if numOfSimAttributes equals 0 then
    similarAttributeFactor = 0
end if
return currentScore * (1 + productGroupFactor) * (1 + similarAttributeFactor^numOfSimAttributes)
```

After algorithms 2 and 3 each product in the database now has an array with the top 10 similar products based on amount of views, description and product group.
Next up is calculating the top products for each visitor, these are the products the specific visitor has viewed the most. This is accomplished by iterating through each visitor, checking their behavior and storing their top products as a field in the database. This calculation can be seen in algorithm 4. This calculation also requires a large amount of resources the first time, but very little to maintain.

Algorithm 4 Calculations of each visitors top products

```
for all Visitors v do
    visitorProducts = Dictionary<string, int>
    for all Behaviors b in v do
        topVistorProducts[b.Id]++
    end for
    Sort visitorProducts after highest value
    Store top 5 visitorProducts in database under v
end for
```

Since all these calculations are made before the actual product recommendations are requested, the process of recommending products is quite fast. The recommendation process starts with retrieving the requested visitor's top products from the database, retrieving these products top similar products, sorting them based on their score and finally returning the amount asked for. The code for the recommendation part can be seen in algorithm 5

Algorithm 5 Get product recommendations

```
visitorTopProducts = db.GetTopProducts(visitorUID)
productRecommendations = Dictionary<int, double>
for all Products p in visitorTopProducts do
    SimilarProducts = db.GetTopProductRecommendation(product)
    for all products simProduct in similarProducts do
        if productRecommendations contains simProduct then
            productRecommendations[simProduct] += similarProducts[simProduct]
        else
            productRecommendations.add(simProduct, similarProducts[simProduct]
        end if
    end for
end for
Sort productRecommendations after highest value
return amount of productRecommendations requested
```

The entire process of requesting product recommendation, running algorithm 5 and returning them takes less than 40ms which is one of the non-functional requirements.

Some other paths are required in certain situations such as when the visitor does not have any behavior or not enough behavior to satisfy the amount of recommendations requested. In these cases the remaining recommendations are filled from the top 20 most popular products in the last 30 days. The top 20 products are calculated by checking the timestamp and finding those in the last 30 days and then counting how many times each product was viewed. The top 20 products are stored in the database and can be calculated through an API call.

3.5 Hosting the API

An API can be hosted in several different ways, through many providers. This product recommendation API is hosted through Amazon WebServices in an *EC2 Instance* [6]. To accomplish this the ASP.NET core project is built in a Docker container, the container is pushed to the Docker Hub and then pulled and run in the *EC2 Instance*. The database is similarly packed in a docker container and run in the *EC2 Instance*. The Docker containers have exposed ports to the rest of the internet and can be accessed via *EC2 Instance* public DNS or IP.

Chapter 4

Experimental Validation

4.1 Validation of recommendations

This section takes two approaches to validating the implemented recommendation engine:

- A statistical approach using recall
- Concrete examples

4.1.1 Statistical approach

Validation of product recommendation engines is focused around two approaches [7]:

- Offline validation
- Online validation

Offline validation

For offline evaluation measures such as Root-Mean-Square-Error (RMSE) and Mean-Absolute-Error (MAE) [8] are often used. These measures requires user ratings on products which is not present in the data for this project. RMSE and MAE can therefore not be used to evaluate these product recommendation.
Another offline method which is used is called Recall. This method functions by using a percentage of the data available as regular input data and another percentage as test data [7]. The Recall evaluation run in this project used 80 percent of the data as input and tested on the remaining 20 percent. More specifically the remaining 20 percent was used in the following way:

- Take each visitor with more than two behaviors
- Input half of the visitor's behaviors via the API
- Generate 5 recommendations for the specific visitor
- See if the remaining half of his behaviors are in the recommended 5 items.

This resulted in a total of 1,934 visitors tested. These visitors have 11,328 behaviors where half was used as input and half was used as control. This means the recommendation engine had to predict half of 11,328 (5,664) behaviors. The algorithm succeeded in correctly predicting 2974 behaviors.
The percentage of how many times the recommendation engine would have predicted a visitors actual behavior is therefore $2974/5664 * 100 = 52.5\%$ the result of this evaluation as well as the scripts used can be found attached with the source code.
The engine only had half of each visitors behavior to go on which could be only 1 behavior and still managed to guess correctly more than half of the time. The 47.5% of the time the algorithm did not correctly guess the behavior does not mean that these were bad recommendations - the visitors

might not have discovered these items themselves but as they were recommended they might have examined these as well. The percentage of correct behaviors guessed would be even higher if more product recommendations where requested in the test.
As the algorithm acquires more data on all visitors as well as the visitor requesting recommendations the predictions will become even more precise.

Online validation

When the recommendation algorithm is put into production several new and better ways of evaluating the system becomes available. As the visitors get recommendations their behavior is logged and it is therefore easy to see how many of the recommendations are actually used and adjust the algorithm thereafter. This adjustment can potentially be automatized by using machine learning.

4.1.2 Concrete examples

To give a better understanding of the recommendations given by the algorithm a few specific examples are given below.

Visitor AAF995AE-1DD0-41C6-898B-9CBEE884E553 has looked at the following products:

- **36991:** Playset Brandmand Sam fyrtårn med figur
- **37691:** Playset Brandmand Sam Havnestation
- **37799:** Firman Sam Ocean Rescue
- **38950:** Sejt Brandmand Sam udstyrssæt
- **40786:** Brandmand Sam helikopter med lys og lyd
- **42373:** Biler Brandmand Sam og brandbil
- **52818:** Udklædning tilbehør sej Brandmand Sam megafon
- **52919:** Playset Fireman Sam

and is recommended the following products:

- **43215:** Biler Brandmand Sam bil
- **36991:** Payset Brandmand Sam fyrtårn med figur
- **37799:** Firman Sam Ocean Rescue
- **38950:** Sejt Brandmand Sam udstyrssæt med bælte
- **34392:** Brandmand Sam 104 cm Fastelavnstøj

The visitor has already looked at some of the items recommended, but others he has not. As the recommendations size increases more new products to the visitor will appear. The recommendations make sense since they are all "Brandmand Sam" stuff which is all he has looked at in the past. In the future the recommendation engine will be able to sort out his orders, but this has yet to be implemented.

Another **visitor 0036ECB4-F5AB-4B7C-824C-8D1C832CB65A** has looked at the following products:

- **43106:** Biler Scalextric C3528 BMW MINI Cooper S
- **49777:** Scalextric Racerbane C1368 Bilbaner Le Mans Prototypes Sports Cars
- **33136:** Chevrolet Camaro GT-R Biler Scalextric C3383
- **43104:** Scalextric C3524 VW Polo WRC Biler

and is recommended the following 5 products:

- **43106:** Biler Scalextric C3528 BMW MINI Cooper S
- **43104:** Scalextric C3524 VW Polo WRC Biler
- **49777:** Scalextric Racerbane C1368 Bilbaner Le Mans Prototypes Sports Cars
- **33136:** Chevrolet Camaro GT-R Biler Scalextric C3383
- **42841:** Maserati Trofeo Biler Scalextric C3388

These recommendations also makes sense theme-wise as they relate closely to the products the visitor has viewed.

A final example **visitor 22C9CF0F-8B96-4764-A2D1-6194992CEDC2** has looked at these products:

- **42809:** Bosch arbejdsbord Bosch Værktøj og Værktøjsbænke
- **42106:** Elsker du også bare paw patrol

and is recommended these products:

- **42106:** Elsker du også bare paw patrol
- **42809:** Bosch arbejdsbord Bosch Værktøj og Værktøjsbænke
- **40542:** LEGO Legends Of Chima Flyv op gennem skyerne
- **31548:** LEGO Legends Of Chima Snurrende slyngplanter
- **43713:** Fastelavnstøj Tid til at ringe efter politiet og Paw Patrols hund nummer 1

The two LEGO recommendations in this example might not seem thematically accurate, however since they have been recommended they must have a high similarity score to one or both of the products the visitor has looked at. The high similarity score is probably due to many visitors having viewed the LEGO products together with the other products in question.

4.2 Conclusion

The system has been validated statistically with the Recall method where the algorithm correctly predicted 52.5% of the visitors' behavior which would be even higher if more recommendations were requested in the test.. Concrete examples have also been examined and the recommendations makes sense when looked at. A few other measures such as RMSE and MAE could not be used due to the lack of product rating from the visitors.

Appendix A

API commands

Function	URI	Example	Description
GET	recommendation/visitorUID/numberOfRecommendations/database	recommendation/AAF995AE-1DD0-41C6-898B-9CBEE884E553/5/Pandashop	Returns a JSON array of size numberOfRecommendations containing productUIDs which are the product recommendations for the specific visitor
PUT	visitor/visitorUID/database	visitor/AAF995AE-1DD0-41C6-898B-9CBEE884E553/Pandashop	Registers a new visitor with the database
PUT	product/productUID/description/productGroup/database	product/5352/Agreatproduct/5/Pandashop	Registers a new product along with its description and product group with the database
PUT	behavior/visitorUID/behaviorType/ItemID/database	behavior/AAF995AE-1DD0-41C6-898B-9CBEE884E553/ProductView/5352/Pandashop	Registers a new behavior for the specific visitor with the database
GET	Update/database/password	Update/pandashop/supersecretpassword	Builds the collaborative filter for the database
GET	Updatevisitortopproducts/database/password	Updatevisitortopproducts/pandashop/supersecretpassword	Updates the top products for all visitors
GET	calculateTop20/database/password	calculateTop20/pandashop/supersecretpassword	calculates the top 20 products in the last 30 days

Bibliography

[1] A. Trivedi. (Feb. 2014). Mapping relational databases and sql to mongodb, [Online]. Available: https://code.tutsplus.com/articles/mapping-relational-databases-and-sql-to-mongodb--net-35650 (visited on 02/23/2017).

[2] M. D. C. Bowen. (Jan. 2017). What is normalized vs. denormalized data?, [Online]. Available: https://www.quora.com/What-is-normalized-vs-denormalized-data (visited on 02/23/2017).

[3] C. Buckler. (Sep. 2015). Sql vs nosql: The differences, [Online]. Available: https://www.sitepoint.com/sql-vs-nosql-differences/ (visited on 02/23/2017).

[4] S. IT. (May 2017). Db-engines ranking, [Online]. Available: https://db-engines.com/en/ranking (visited on 05/07/2017).

[5] G. L. B. Smith and J. York. (Feb. 2003). Amazon.com recommendations: Item-to-item collaborative filtering, [Online]. Available: https://www.cs.umd.edu/~samir/498/Amazon-Recommendations.pdf (visited on 05/08/2017).

[6] Amazon. (May 2017). Getting started with amazon ec2 linux instances, [Online]. Available: http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/EC2_GetStarted.html (visited on 05/08/2017).

[7] T. Řehořek. (Dec. 2016). Evaluating recommender systems: Choosing the best one for your business, [Online]. Available: https://medium.com/recombee-blog/evaluating-recommender-systems-choosing-the-best-one-for-your-business-c688ab781a35l (visited on 05/09/2017).

[8] J J. (Mar. 2016). Mae and rmse which metric is better?, [Online]. Available: https://medium.com/human-in-a-machine-world/mae-and-rmse-which-metric-is-better-e60ac3bde13d (visited on 05/09/2017).