Optimization and personalization of a web service based on temporal information

Jonatan Wallin (jova0034@student.umu.se)

June 25, 2018
Abstract

Development in information and communication technology has increased the attention of personalization in the 21st century and the benefits to both marketers and customers are claimed to be many. The need to efficiently deliver personalized content in different web applications has increased the interest in the field of machine learning. In this thesis project, the aim is to develop a decision model that autonomously optimizes a commercial web service to increase the click through rate. The model should be based on previously collected data about previous usage of the web service. Different requirements for efficiency and storage must be fulfilled at the same time as the model should produce valuable results.

An algorithm for a binary decision tree is presented in this report. The evolution of the binary tree is controlled by an entropy minimizing heuristic approach together with three specified stopping criteria. Tests on both synthetic and real data sets were performed to evaluate the accuracy and efficiency of the algorithm. The results showed that the running time is dominated by different parameters depending on the sizes of the test sets. The model is capable of capturing inherent patterns in the available data.
Acknowledgement

I would like to thank the people at Codemill AB for giving me the opportunity to writing this master’s thesis with them. A special thanks to my external supervisor Emil Lundh for giving me advices and support through out this project. I am also grateful to my examiner Eduardo Gracia for giving me valuable inputs and guidance.
Contents

1 Introduction 1
  1.1 A short history of advertising . . . . . . . . . . . . . . . 1
  1.2 Online advertising . . . . . . . . . . . . . . . . . . . . . . 2
    1.2.1 Online advertising formats . . . . . . . . . . . . . . . 2
    1.2.2 Personalization . . . . . . . . . . . . . . . . . . . . . 3
  1.3 Workplace . . . . . . . . . . . . . . . . . . . . . . . . . . . 3
  1.4 Background . . . . . . . . . . . . . . . . . . . . . . . . . . 4
  1.5 Project aim . . . . . . . . . . . . . . . . . . . . . . . . . . 5
  1.6 Requirements . . . . . . . . . . . . . . . . . . . . . . . . . 5

2 Theory 7
  2.1 Information theory . . . . . . . . . . . . . . . . . . . . . . 7
  2.2 Machine Learning . . . . . . . . . . . . . . . . . . . . . . . 9
  2.3 Discretization . . . . . . . . . . . . . . . . . . . . . . . . . 12

3 Method 19
  3.1 The problem as a machine learning task . . . . . . . . . . . . 19
  3.2 Representation of the temporal data . . . . . . . . . . . . . . 19
  3.3 Partitioning the dataset . . . . . . . . . . . . . . . . . . . . 21
  3.4 Procedure . . . . . . . . . . . . . . . . . . . . . . . . . . . 23
  3.5 Stopping criteria . . . . . . . . . . . . . . . . . . . . . . . . 24
  3.6 Algorithms . . . . . . . . . . . . . . . . . . . . . . . . . . . 25
  3.7 The pipeline . . . . . . . . . . . . . . . . . . . . . . . . . . 27

4 Results 28
  4.1 Performance . . . . . . . . . . . . . . . . . . . . . . . . . . 28
  4.2 Tests on artificial datasets . . . . . . . . . . . . . . . . . . 31
  4.3 Tests on real datasets . . . . . . . . . . . . . . . . . . . . . 34

5 Discussion 37
  5.1 Future work . . . . . . . . . . . . . . . . . . . . . . . . . . 38
List of Figures

1. An example of a Smart Video. ........................................... 4
2. The graph shows how the entropy depends on the probability $p$
in a coin toss. .............................................................. 8
3. Polynomials fitted to data points. .................................... 10
4. Polynomial fitted to data points. ..................................... 12
5. The available cut points of a data set. ......................... 13
6. The figure shows the existing boundary points a toy set. .... 14
7. A cut point $T$ within a group of $n_j$ examples of the same class. 15
8. Figure of product clicks with two different playlists. ........... 20
9. The product clicks projected onto the unit circle. ............... 20
10. A partition obtained by using a two cut point search. ......... 22
11. Running time versus the set size. .................................... 28
12. The number of boundary points depend linearly on $|S|$. ...... 29
13. Runtime versus set sizes in a larger interval. ................. 30
14. The number of boundary points approaches a constant. .... 30
15. The runtime increases more rapidly with the flexible approach. 31
16. The result of created dataset when the examples can be purely
    separated. ................................................................. 32
17. The algorithm finds three pure sets and isolates the mixed regions. 33
18. The effect of the $ent_{\min}$ parameter. ......................... 34
19. The resulted partition of a small dataset with two playlists. . 35
20. A real world dataset with 20 000 product clicks. The partition is
    finer in the period Thursday - Saturday because the majority of
    the clicks occurred in that time period. ......................... 36
1 Introduction

The Internet, the largest existing network, was intended to increase the accessibility to information for people in its early history [1]. After the World Wide Web was introduced in the 1990’s the platform was converted from an individual environment to a commercial one with almost three million web pages by 1994, where 89% originated from commercial companies [2]. The rapid growth and popularity of the World Wide Web has not been seen in any traditional media, such as radio or TV. It took only 5 years to reach 50 million users and nowadays the Internet is part of almost everyone’s life and constitutes the main platform for consumption [2].

1.1 A short history of advertising

Even though modern advertising differs remarkably from the way it was used in its early days, advertising has been part of most societies back to the ancient cities of Babylon, Sumer and Jerusalem [2]. Back then, advertisement took expression in its simplest form. There was a crier announcing the goods available from local traders to catch people’s eyes. Furthermore, outdoor advertisement (today’s banners and neon signs) began in ancient Egypt and was used in various ways to deliver information and messages to a wider range of audience. A parallel to the sport advertisement in modern societies could be seen in the use of big placards in the gladiator combats in the Roman empire [3]. Handbills were developed in the 1400s, and in the 1600s outdoor signs appeared in the cities which later, with the contribution of Edison, transformed into neon signs lighting up the nights in various cities.

In the seventeenth century, when newspapers entered the society, the form of advertisement we are familiar with started. Even though the content of those announcements is not what one would expect in ads today, the principle was the same. Along with the progression made in the field of communication technology, radio was available to most households in the US in the 1920s. With its big audience it was a great opportunity to effectively introduce new products in the market and make people aware of brand names [3].

A time after the second world war, television came to be the main broadcasting medium and with that a new opportunity for commercial purposes. In Britain 1955, a mix of commercial, financial and political interests resulted in the fall of BBC’s monopoly of broadcasting which opened up the door for commercial television [4]. In the US, the first TV commercial appeared in 1941 just before a baseball game when a company paid 9$ for a 10 second spot on the screen. In 2005, the number of TVs in worldwide was estimated to 1.7 billion and thus constituted as one of the largest parts of the total advertising revenue. The investments in advertising have since then increased drastically during the years. For example, in 2007 an estimated 279 billion dollars was spent only in the US [2,3].
1.2 Online advertising

The purpose with online advertising is similar to advertising in traditional media: Disseminate information to people with the aim to affect a buyer-seller transaction. However, there is an important difference, the Internet provides the possibility for consumers to interact with the commercial content. On the other hand, the advertisers can thus collect information about the users which was not possible before [5]. Furthermore, online advertising offers a much shorter path from an ad impression to an actual transaction than any other media we have witnessed before, only a few clicks are required to initiate a purchase [1].

There are several different stakeholders within the online advertising industry. Broadly speaking, the industry can be divided into three parts including the sellers, buyers and the infrastructure. The sellers are publishers owing websites that attract audience and profit from selling ad space on these sites. The buyers are the companies having products/services they want to promote by online advertising and thus have a demand for the available ad spaces. The link between the seller and buyers are the advertising infrastructure whose purpose is to create tools to deliver the ads and make online advertising possible [1].

1.2.1 Online advertising formats

There are diverse online advertising formats. The most commonly used methods are banners, which are graphical elements usually placed at the top of a website with the intention to transfer the visitor to the publisher’s site by clicking on the object. Due to a reduction of the click-through rates (ratio between page views and clicks, commonly denoted as CTR) of banners in the early 2000’s, pop-up ads were invented as a new alternative. However, nowadays there are diverse software available which can prevent them from appearing in the users’ screens [2].

When email emerged as a new communication method the advertisers saw the opportunity to promote their services and products using this new channel. The only problem was the need of the email addresses of potential customers. Many advertisers solved this issue by offering ”free access” or other kinds of benefits if the user provides personal information, including the email address, which later can be used for commercial purposes [2].

The number of web pages have been rapidly increasing since the creation of the Web, making it necessary with tools for traversing the jungle of web sites in order to present relevant websites to a user request. These tools are what we know as search engines today. It was soon discovered that there exists a lucrative market within the web-search industry. Search engines have the advantage to direct the ads to a public likely interested in the product/service. In some sense, it is not a bad assumption that search engine queries supplied by the user can be related to the intent/interests of the user self. Thus, advertisements somehow related to the query are likely to catch the user’s attention [6].

When video broadcasting started to be transferred from television to the Internet, online video advertising emerged as a natural consequence [2]. Video ad-
advertisement online can either offer the viewer playback control functions (such as play and pause) or videos without them. The former usually constitutes independent videos whereas the latter format is usually embedded in other online videos [7]. Today, advertising in online videos is a billion dollar industry and the investments are predicted to increase in the upcoming years [8].

1.2.2 Personalization

As mentioned before, the Internet is an interactive medium allowing companies and publishers to collect information about the visitors. This information was proven to be very valuable to advertisers since it can be used to customize ads. The usage of customer information to tailor commercial content to individual costumers is called personalization, even though this is just one out of many other definitions of the concept [9]. Development in information and communication technologies has increased the attention of personalization in the 21st century and the benefits to both marketers and customers are claimed to be many [10]. From the marketers’ point of view, investments on ads that are shown to people not likely to positively react on them can be seen as a bad move, while the customer experience is deteriorated by targeting people with irrelevant ads since it can be perceived as spam.

The personal data that is available to the marketers, and thus can be used for personalization, have different origins. A study performed in 2016 surveyed the sources of this kind of data [11]. The results revealed that some of the most common sources were purchasing history, socio-economic indicators, information about networks/devices used, media consumption, activities on social media sites, public records and user supplied surveys. Real time data such as IP-address and date-time is already available in the header of the HTTP-request at a web page visit.

The huge Internet traffic require fast processing of data in order to present personalized content effectively [13]. Therefore, algorithms and models that handle data automatically are necessary and the interest in the field of machine learning has growth. The traces left by customers traversing the Web can then together with these algorithms automatically customize commercial content on the web and mobile applications.

A survey from 2017 showed that consumers have a more negative rather than positive attitude towards the concept of personalization [12]. The resistance seemed to originate from privacy concerns together with the relevance of the experienced personalized ads. In other words, the resulting personalized ads are valued less compared to the consumers’ perceived privacy violations.

1.3 Workplace

The project took place in Umeå at the company Codemill AB\(^1\) which is a software development company specialized in video technology and digital product development. The company was founded in 2007 and today it consists of around

\(^1\)https://codemill.se/
50 employees. Besides working with local customers, they have a focus on the international market and have established partnerships with big international companies mostly in the media and fashion industry. With an origin at Umeå University the company is still closely connected to the academic world.

1.4 Background

One of the products developed by Codemill is Smart Video, and today it is also an affiliate to the company. Smart Video offers interactive video solutions for the users and the aim is mainly to help brands and influencers to increase their online sales by adding more value to their commercial videos. The CTR can be increased up to a factor of 10 with this technology\(^2\). Figure 1 shows one example of a SmartVideo.

\begin{figure}[!h]
\centering
\includegraphics[width=\textwidth]{smart_video.png}
\caption{An example of a Smart Video. In the video one can see an influencer doing a make up tutorial. The product cards in the product bar are activated when the corresponding product is used in the video. A visitor will be redirected to a website, where the product is available for purchase, by clicking on a product card.}
\end{figure}

The idea is to extend the regular advertising videos on their web pages by including interactive product cards showing up at specific times when the corresponding products are seen in the video. In the upper section an advertising video (1) is played while a product bar (2) is placed in the lower part of the player. The product bar contains a playlist of the product cards (3) representing

\(^2\)https://smartvideo.io/
the promoted products in the video. The video player is designed such that the focus should be primary on the video content and secondary on the promoted products which is one of many reasons to the success.

By clicking on a product card the visitor will be redirected to the corresponding web page where the product can be ordered online. The product cards are arranged, by the user, in a playlist and one specific video can have several playlists that can differ in style and included product cards etc. Even though Smart Video is primarily used for advertising purposes, it is used in other applications as well. This thesis project will hopefully contribute to make the Smart Video platform even smarter.

1.5 Project aim

The goal of the project is to implement a decision model that autonomously optimizes the Smart Video player based on available information about a visitor. I will refer to a person that has created a Smart Video as a user and a person entering a site where a Smart Video is present as a visitor. Examples of available data about a visitor are geographical data, date and time of the visit together with other information such as web browser and device can also be obtained. Information in customer files can also be used as long as it is within the law and the customers have approved the usage.

The optimization, in this context, means deciding within a predefined time which of the created playlists that should be used in the current session. The hypothesis is that different kinds of visitors/visits have different preferences and the model should predict these preferences based on previously collected data. Personalization like this hopefully results in an increase in online-sales for the users while the visitor will find the content shown more relevant and therefore a contribution to a better experience.

In this project temporal information is the data of interest. Temporal information is information about time and when a visitor enters the web page a time stamp, consisting of date and time, is stored in a database together with an event. The events could be a page-load or a product click as two examples. The ratio between the number of page-loads and product click defines the click through rate (CTR) in this case. The model should therefore choose which playlist will be used in the current session based on the time stamp, with the goal to increase the overall CTR of the commercial video. Since new data are available through new visits of a web page with a Smart Video, the model will be updated daily and thus needs to be self-optimized.

1.6 Requirements

Temporal information about a visit is available once the visitor loads the page. Thus, the time between the page load and once a decision is made by the model is a critical factor. Recommendations from the Interactive Advertising Bureau (IAB) states that personalized ads should be rendered within 300 ms [14].
Therefore, the model should be able to make a decision not exceeding this limit to ensure performance. The memory needed to store the model cannot exceed the Megabyte scale.

The expected outcome is an implementation of the decision model in Python that can be integrated in the existing code. Two modules must be part of the final implementation. One concerning the building of the model given data from a database where statistics are gathered, referred as the building module. This module will hold the rules that are used when decisions are made. The other part handles the decision process given the time stamp of a visit and is called the search module. The search module will be operating on the rules generated by the building module. The building module will be rebuilt when the decision model should be updated while the search module will be used for every visit of a web page where Smart Video is present. Since the model will be rebuilt at most a few times a day the building module is not as time critical as the search module but should be polynomial in time complexity to allow scaling.
2 Theory

2.1 Information theory

Information theory is a field of science where quantification, storage and communication of information are of interest. In communication theory it answers questions about limits in data compression and what the ultimate rate of transmission of data is. Many other fields are also related to information theory and it has made contributions to physics, economics, statistics and computer science, among others [15]. The origin of the theory was the paper A Mathematical Theory of Communication written by Claude Shannon in 1948 [16]. The publication of that paper started the evolution of information theory.

Shannon considered how a quantity could be defined to measure the uncertainty in a discrete random variable. Suppose we have a discrete random variable $X$ with an alphabet $\mathcal{X} = \{x_1, x_2, \ldots, x_n\}$ and a probability mass function $p_X(x) = Pr(X = x)$, $x \in \mathcal{X}$. We define $p_i \equiv p_X(x_i)$ to be the probability of the outcome $x_i$. The quantity describing the uncertainty in $X$ should possess three reasonable properties according to Shannon. We call the quantity $H$, then these requirements should hold:

1. $H$ should be continuous in the probabilities $p_i$.

2. If $|\mathcal{X}| = n$ and the probabilities are uniformly distributed such that $p_i = \frac{1}{n}, \forall i$, then $H$ should be a monotonic increasing function of the number of possible events $n$. This is reasonable since when introducing more possible outcomes the uncertainty should strictly increase.

3. If a choice is broken down into two successive choices, then the original quantity should be the weighted sum of the new individual values. To exemplify this, consider three events $\{x_1, x_2, x_3\}$ with probabilities $p_1 = \frac{1}{2}, p_2 = \frac{1}{3}, p_3 = \frac{1}{6}$. The event $x_1$ occur with probability one half and the event that any of $x_2$ or $x_3$ occurs also with probability one half ($\frac{1}{3} + \frac{1}{6} = \frac{1}{2}$). Thus, the original choice between all the events could be broken down into two successive choices where the first choice is if it is $x_1$ or any of $x_2$ or $x_3$. If the answer is the latter, the choice is between $x_2$ or $x_3$. Mathematically, the formula $H\left(\frac{1}{2}, \frac{1}{3}, \frac{1}{6}\right) = H\left(\frac{1}{2}, \frac{1}{2}\right) + \frac{1}{2}H\left(\frac{1}{3}, \frac{1}{3}\right)$ should hold. The coefficient $\frac{1}{2}$ in front of the last term is present because the last choice need only be made half of the times.

He shows that the $H$ needs to be of the form

$$H = -K \sum_{x \in \mathcal{X}} p_X(x) \log(p_X(x)), \quad (1)$$

where $K$ is a positive constant, to establish the requirements above. The choices of $K$ and the base of the logarithm determines the unit of measure. One thing to notice is that equation (1) is very similar to Gibbs entropy formula in statistical physics where $K$ is set to Boltzmann’s constant, the natural logarithm is used, the sum (integral in the continuous case) is taken over the allowed states in the phase space in the physical system and $p_i$ is the probability of finding the system is state $i$ [17]. In physics, entropy is commonly referred to as a quantity
describing the disorder in a system.

When the constant is set to unity and the base of the logarithm is set to 2, \( H \) is called the *Shannon entropy* (I will refer to this as just entropy from now on) and is measured in bits

\[
H = -\sum_{i=1}^{n} p_i \log_2(p_i). \tag{2}
\]

The formula in (2) does not depend on the specific values that the random variable can assume, instead it is a functional of the probability distribution inherent in the random variable. Further, it is always greater or equal to zero since \( 1 \geq p_i \geq 0 \). One could think that a problem arises if some \( p_i = 0 \), since the logarithm diverge to negative infinity. Using the convention that such a term is zero can be justified by the limit

\[
\lim_{x \to 0^+} -x \log(x) = 0. \tag{3}
\]

The convention also make sense since adding events with probability zero will not increase the uncertainty. To illustrate the behaviour of the entropy we can consider a coin flip with the outcome "head" with probability \( p \) and outcome "tail" with probability \( 1-p \). Plotting the entropy as a function of the probability \( p \) would yield the graph seen in Figure 2

![Figure 2](image)

**Figure 2**: The graph shows how the entropy depends on the probability \( p \) in a coin toss. The outcome "head" happens with probability \( p \) and "tail" with probability \( 1-p \). When \( p = \frac{1}{2} \) the entropy is maximized.
In the extreme cases, $p = 0$ and $p = 1$, the entropy tends to 0 since there are no uncertainty inherent in the stochastic process, and in fact the random variable is not random at all. As observed, the entropy takes on its maximum value at $p = \frac{1}{2}$ in a fair coin toss when the uncertainty is maximized intuitively.

### 2.2 Machine Learning

The sub-field of *artificial intelligence* that considers how programs/models learn from experience is called *machine learning*. Usually, input-output pairs are supplied to the model, from which a general function should be learned to predict outputs from previously unseen input data. The available feedback from these input-output pairs distinguish three main types of learning. In *unsupervised learning* the model learns patterns in the input data without any given output. One task of this kind commonly encountered is *clustering* [18]. When the feedback comes as a series of rewards/punishments as a response to the action of a learning agent it is called *reinforcement learning*. In the cases where there are available example data containing input-output pairs and the model should learn a function that maps the inputs to the outputs, it is called *supervised learning* [19]. The rest of this subsection will focus on supervised learning.

The general setting in a supervised learning task consist of a dataset of $N$ examples where each example is a pair which includes input data together with the corresponding desired output data. The examples are commonly denoted as $(x_1, y_1), \ldots, (x_N, y_N)$ where the different $x_i$:s are the input data while the output data is labelled $y_i$. Each $y_i$ is supposed to be the output of an unknown function $f(x)$ and the aim is to find a hypothesis function $h$ that approximates the true one well [19]. The true function $f$ does not necessarily need to be a deterministic function, it can also be a stochastic one.

The outputs are called *targets* and can either be quantitative or qualitative. When these variables are quantitative they take on numerical values, such as a stock price, and the learning problem is referred to as a *regression problem*. If the outputs are qualitative they take on values from a finite number of different *classes*, such as animal species, and the problem is called a *classification problem*. Whether a task is a regression or classification problem often affects the choice of applied machine learning method even though there are models that handles both cases [19]. However, many machine learning methods used for classification can be probabilistic which means that the models predict the probabilities that the input belongs to each different class. Since probabilities are numerical values this could also be interpreted as regression which shows that the difference between classification and regression problems is very subtle.

The input variables $x_i$ are called *features* and, like the predictors, the features can also be quantitative or qualitative. When there are more than one feature the $x_i$:s turn into vectors instead. Often, one of the main components in a machine learning task is to, in some way, know which features are of importance and which ones are not. Sometimes two or more features are correlated, which can be problematic in models where feature independence is assumed. Methods for feature selection have been developed and are simply used as preprocessing
Least squares is one simple parametric model used for regression problems [20]. It is used in situations where fitting curves to observed data points are of interest. The idea is to learn a function from the available data points in order to make predictions of unseen data. Before proceeding with the fitting procedure, one has to define a hypothesis space from where to search for the hypothesis function $h$. The space could be all polynomials of degree 1 or all polynomials of degree 6 as two simple examples. Then, the problem reduces to find the coefficients that specify the exact representation of $h$ minimizing the error $\|f - h\|_2^2$.

When deciding the hypothesis spaces, one also restricts the flexibility of the shape of the available functions. By returning to the examples of the two polynomial spaces, the one of degree 1 is less flexible than the one of degree 6 (there are more coefficients in an arbitrary degree 6 polynomial than an arbitrary linear polynomial) [17]. Figure 3 shows an example of two least square fits of simulated data points from the equation

$$y = 2x - 1 + \epsilon,$$  \hspace{1cm} (4)

where $\epsilon \sim N(0, \frac{1}{2})$ is a noise term from a Gaussian distribution with expectation value 0 and standard deviation $\frac{1}{2}$.

![Figure 3](image)

**Figure 3:** The graph shows 7 data points together with two curves obtained with the use of a least square fitting procedure. The solid line is the result when coefficients for a degree 6 polynomial are calculated and the dashed line is a degree 1 polynomial.

As seen in the figure above the degree 6 polynomial fits the data points almost
perfectly. This curve would yield an error close to zero while the linear model would result in a larger error. At first glance one might think that the more flexible curve is the best one since the error is effectively zero, but after a while the conclusion would be that this curve does not seem to generalize well on unseen input data. By simply observing the raw data points, it is possible to observe a trend where $y$ increases as $x$ increases, however at the right most side of the solid curve a rapidly decreasing behaviour can be seen. At the same time the linear model follows the trend nicely and as noted before the data points were simulated from a linear model.

The above scenario is a well recognized phenomenon in the field of machine learning and is known as overfitting. The essence is that in theory one can make a model fit observed data points almost exactly in most cases by choosing a flexible enough hypothesis space such that the error of the data points in the training set is very close to zero. Because of the high complexity of the model the curve could look completely different if one would simulate a new set of points using the same process, thus the model has a high variance and is not likely to generalize well for unseen data [21].

In the case of the linear model, the change in the shape of the curve would not be as large if new points were simulated, instead the shapes would be closely related but the result would still have an error rate of importance. The above discussion is related to the so called Bias-variance trade off and need to be considered when building a machine learning model. Variance is increased and bias decreased when the model complexity grows and vice versa [21]. In Figure 4 new points are simulated using equation (4) with Gaussian noise as before and two new curves are obtained.
As seen above the linear curve looks almost the same as in Figure 3 while the degree 6 polynomial looks completely differently.

### 2.3 Discretization

Consider a set $S$ of $N$ examples with $k$ different classes and a continuous feature $F$. Let the set of values $F$ can attain be denoted as $\text{range}(F)$. A threshold value $T$ that discretizes the range of the continuous feature into two subintervals is called a *cut point*. The subintervals are induced by $T$ are $[\text{min}(\text{range}(F))...T]$ and $(T...\text{max}(\text{range}(F))]$. The original set of examples are then partitioned into two subsets $S_1, S_2$ according to the tests

$$
S_1 = \{ e \in S | e(F) \leq T \} \\
S_2 = \{ e \in S | e(F) > T \}
$$

where $e(F) \in \text{range}(F)$ is defined to be the value of the feature $F$ for a given example $e$. Assuming that $e(F) \neq e'(F) \quad \forall e, e' \in S | e \neq e'$ the examples can be sorted with respect to the feature in increasing order without anyone sharing the same value of $F$ and thus we can choose between $N-1$ different cut points to discretize $S$. Each candidate cut point is a point between each successive pair in the sorted sequence of numbers. Figure 5 illustrates the available cut points with a set of 12 examples from two different classes. The red lines mark...
the different partitions that can be made.

![Figure 5: The figure shows an example of a set where two different classes exist. All available partitions are illustrated with the red lines. The cut points are the mid points between each successive pair when sorted with respect to feature $F$.](image)

But which of all the available points should be chosen? A solution is to measure the quality of a candidate cut point. One possibility, commonly used, is derived from the definition of entropy in equation (2) [22]. One can think of the classes in the examples in $S$ to be outcomes from a random variable, with a probability distribution over the interval range($F$). When a cut is performed, the set $S$ is partitioned into two subsets $S_1$ and $S_2$ (according to the two tests stated above) the entropy of each subset can be calculated. This is done by approximating $p_i$ to be equal to the proportion of each class within the subset. The quantity is called class entropy. By labelling the $k$ classes, $C_1, ..., C_k$ and the proportion of examples having $C_i$ as class $P(C_i)$, the class entropy of a set $S$ is defined as

$$\text{Ent}(S) = -\sum_{i=1}^{k} P(C_i) \log_2 (P(C_i)).$$

Similarly as the reasoning about equation (2), it can be concluded that equation (6) is minimized if the set is totally pure (only contains examples from one single class) and maximized if there are equally many examples from each class, i.e. $P(C_i) = \frac{1}{k}$.

After splitting the original set into two subsets the class entropy of each subset can be calculated. The quantity

$$E(F,T; S) = \frac{|S_1|}{|S|} \text{Ent}(S_1) + \frac{|S_2|}{|S|} \text{Ent}(S_2),$$

is called class information entropy of the partition induced by $T$, where $F$ indicates that the set has been partitioned with respect to the feature $F$. Observing that (7) is the weighted sum of the entropies of the subsets where the weights are proportional to the sizes of these two new sets. This makes large sets more important compared to smaller ones. The introduction of the weights is important since one can always do a partition resulting in one totally pure set by
isolating one example in one subset and all the other \(N - 1\) examples in the other subset. Since the aim of the discretization is to find inherent structure/patterns in the data, such small sets is not desirable [22]. The cut point that minimizes (7) is chosen by evaluate the quantity for all possible cut points. The resulting subsets can then be recursively partitioned with the same procedure until a pre-defined stopping criterion is met.

*Information gain* is defined as

\[
I(T; S) = \text{Ent}(S) - E(F, T; S),
\]

so by minimizing (7), the information gain is maximized since \(\text{Ent}(S)\) is constant for a given set of examples.

Fayyad shows in [23] that the minimization of (7) can only occur on a *boundary point*. If \(e_i \in S\) represent an example in the set and the feature value of that example is denoted by \(e_i(F)\) we can define a boundary point as follows:

**Definition 1:** A value \(T\) is a boundary point if and only if there exists two examples \(e_1, e_2 \in S\) of different classes when the set is sorted with respect to feature \(F\), such that \(e_1(F) < T < e_2(F)\); but there exists no \(e_3 \in S\) such that \(e_1(F) < e_3(F) < e_2(F)\).

In other words, a value is a boundary point if it lies between two consecutive examples from different classes when the set is sorted with respect to a feature \(F\). Thus, 5 candidate cut points can be removed from the toy set illustrated in Figure 5 when searching for the point minimizing (7). The boundary points in the same toy set can be seen in Figure 6. In the special case when two or more examples share the same value of the feature and belong to two or more classes, they cannot be separated. However, one must treat the cut points on either side of such a group as boundary points [23].

![Figure 6](image_url)

**Figure 6:** The figure shows the existing boundary points in the toy set previously shown in Figure 5.

A summary of Fayyad’s proof is presented here. Consider a set with \(S\) of \(N\) examples with classes \(C_1, ..., C_k\) and assume that the set is sorted with respect
to a feature $F$ in increasing order. Imagine the scenario that the cut point $T$ occurs within a group of $n_j \geq 2$ examples from the same class (not a boundary point according to definition 1), particularly suppose it is class $C_k$. Within this group of $n_j$ examples, assume there are $n_c$ many that have a value of $F$ less than $T$. Hence, the inequality $0 \leq n_c \leq n_j$ must hold.

Define $T_1$ and $T_2$ to be boundary points such that the group with the $n_j$ examples have a value of $F$ greater than $T_1$ but less than $T_2$. There are $L$ examples in the set $S$ with value less than $T_1$ and $R$ examples having a value greater than $T_2$. The set up can be seen in Figure 7.

Figure 7: A cut point $T$ within a group of $n_j$ examples of the same class.

Of the $L$ examples to the left of $T_1$ in Figure 7, let $L_i$ be the proportion of $C_i$. Similarly let $R_i$ be the proportion examples of class $C_i$ of the $R$ examples to the right of $T_2$. Then we have the following three relations

$$\sum_{i=1}^{k} L_i = L, \quad (9)$$

$$\sum_{i=1}^{k} R_i = R, \quad (10)$$

$$L + R + n_j = N. \quad (11)$$

The location within the group of $T$ depends on the value of the variable $n_c$. By the definitions of the introduced variables above, we can expand the first term in (7) when we choose $T$ as cut point in the described set up above. Using relations 9, 10 and 11 we can make the following expansion
\[
\frac{|S_1|}{|S|} \text{Ent}(S_1) = - \frac{L + n_c}{N} \sum_{i=1}^{k-1} \frac{L_i}{L + n_c} \log_2 \left( \frac{L_i}{L + n_c} \right) \\
+ \frac{L_k + n_c}{L + n_c} \log_2 \left( \frac{L_k + n_c}{L + n_c} \right)
\]

\[
= - \frac{1}{N} \left[ \sum_{i=1}^{k-1} L_i \log_2 \left( \frac{L_i}{L + n_c} \right) + (L_k + n_c) \log_2 \left( \frac{L_k + n_c}{L + n_c} \right) \right]
\]

\[
= - \frac{1}{N} \left[ \sum_{i=1}^{k-1} L_i \log_2 (L_i) - \log_2 (L + n_c) \sum_{i=1}^{k-1} L_i \\
+ (L_k + n_c) \log_2 (L_k + n_c) - L_k \log_2 (L + n_c) \\
- n_c \log_2 (L + n_c) \right]
\]

\[
= - \frac{1}{N} \left[ \sum_{i=1}^{k-1} L_i \log_2 (L_i) - L \log_2 (L + n_c) \\
+ (L_k + n_c) \log_2 (L_k + n_c) - n_c \log_2 (L + n_c) \right]
\]

\[
= - \frac{1}{N} \left[ \sum_{i=1}^{k-1} L_i \log_2 (L_i) + (L_k + n_c) \log_2 (L_k + n_c) \\
- (L + n_c) \log_2 (L + n_c) \right].
\]

In a similar way we can also derive an explicit expression for the second term in (7),

\[
\frac{|S_2|}{|S|} \text{Ent}(S_2) = - \frac{1}{N} \left[ \sum_{i=1}^{k-1} R_i \log_2 (R_i) + (R_k + n_j - n_c) \log_2 (R_k + n_j - n_c) \\
- (R + n_j - n_c) \log_2 (R + n_j - n_c) \right].
\]

Introducing the function \( G(x) = x \log_2 (x) \) and using equation (12) as well as (13), we can conclude that

\[
E(T, S) = \frac{1}{N} \left[ - \sum_{i=1}^{k-1} G(L_i) - G(L_k + n_c) + G(L + n_c) - \sum_{i=1}^{k-1} G(R_i) \\
- G(R_k + n_j - n_c) + G(R + n_j - n_c) \right]
\]

Observing that \( \frac{d}{dx} G(C + x) = \log_2 (C + x) + \frac{1}{\ln(2)} \) where \( C \) is a constant, and similarly \( \frac{d}{dx} G(C - x) = - \log_2 (C - x) - \frac{1}{\ln(2)} \), we can differentiate expression
w.r.t $n_c$ to see where $T$ should be located in order to minimize (7). The final expression is

$$
\frac{d}{dn_c} E(T, S) = \frac{1}{N} \left[ \log_2 (L + n_c) - \log_2 (L_k + n_c) - \log_2 (R + n_j - n_c) + \log_2 (R_k + n_j - n_c) \right].
$$

(15)

By taking the second derivative w.r.t $n_c$ we arrive at

$$
\frac{d^2}{dn_c^2} E(T, S) = \frac{1}{N} \left[ \frac{1}{L + n_c} - \frac{1}{L_k + n_c} + \frac{1}{R + n_j - n_c} - \frac{1}{R_k + n_j - n_c} \right].
$$

(16)

Recalling that $n_c \leq n_j$, $L_k < L$ and $R_k < R$ the conclusion is that $\frac{d^2}{dn_c^2} E(T, S) < 0$ for $0 \leq n_c \leq n_j$ so the minimum of $E(T, S)$ must be achieved at some of the edges of the interval. Since both $n_c = 0$ and $n_c = n_j$ are boundary points by definition, this shows that the minimizer to $E(T, S)$ can only be a boundary point. □

Remark that the assumption of the group belonging to the particular class $C_k$ does not affect the validity of the proof, since the derivation is possible assuming any other class as well. The choice of $C_k$ only simplifies the notion when dealing with the sums in the expansion of the terms in (7).

Fayyad’s result has at least two positive effects on the discretization procedure. It never favours cut point that will unnecessarily separate examples from the same class because of the boundary point property shown above. The efficiency is also improved since the number of candidate cut point is reduces because of the boundary point restriction.

In one extreme case there will only be $k - 1$ available boundary points, if all the examples from the same class are adjacent in the sorted sequence. In the other extreme scenario, every cut point is a boundary point. That happens when the classes change from each example in the sequence, hence there will be $N - 1$ candidates. Particularly, consider a set of $n$ examples from $k$ classes and let $b(n)$ denote the expected number of boundary points when the set is sorted with respect to a specific feature. Assuming all classes are equally likely (probability $\frac{1}{k}$) and that there is no correlation between the value of the feature and the class of an example we can conclude that

$$
b(1) = 0,
$$

(17)

$$
b(2) = 1 - \frac{1}{k} \cdot \frac{k - 1}{k},
$$

(18)

and in general we have
\[ b(n) = b(n - 1) + \frac{k - 1}{k} = b(n - 2) + 2\frac{k - 1}{k} = \ldots = (n - 1)\frac{k - 1}{k}. \] 

(19)

If there were two classes, the expected number of boundary points would be \( \frac{n - 1}{2} \) which is half the number of available cut points. The assumptions above do not apply to real-world datasets in most cases. Specifically, assuming there is no correlation between a feature and a class does not hold in general since in a machine learning task because the involved features are chosen just because of the assumption that there are some inherent correlation with the classes. Thus, equation (19) can be considered as a worst case and in real-world dataset the number of boundary points are probably even fewer.
3 Method

As stated in section 1.5, the aim is to find a method for choosing the playlist that should be used in a Smart Video given the weekday and time of the day. Time stamps for events such as page loads or product clicks are stored in a database together with an id of the playlist shown in the event. The temporal information (the time stamps) is represented as dates including the time when the events occurred. Afterwards, the data is downloaded and stored in a file of json-format. In this problem the product click events are the data of interest.

3.1 The problem as a machine learning task

Making decisions based on previously collected data sounds very much like a machine learning problem according to the descriptions in section 2.2. The examples consist of dates and times of product-clicks together with the ids of the specific playlists that were used in the events. Thus, the feature(s) is the temporal information about the product-clicks and the predictor is the id of the playlist. The problem can then be interpreted as a supervised learning problem with the aim to predict what playlist should be shown given the weekday and time when a visitor enters a web-page with a Smart Video. This should be done to increase the CTR of a particular video. Since the predictors are qualitative it is a classification task where the classes are the available playlists.

3.2 Representation of the temporal data

To represent time data, as weekday and hour of the day, there are essentially two intuitive options. The first one is to work with two different features, one represents the weekday and the other the hour of the day. Alternatively, one can describe both weekday and time implicitly by using only one feature that represents the time passed since the week start (Monday 00:00). By specifying the time elapsed since the week start one uniquely specifies the weekday and hour of the day at the same time. The single feature can be expressed in any time scale such as seconds, minutes or hours depending on the desired level of precision. If measured in minutes, the feature would take on the value 0 minutes at Monday 00:00 and 10 079 minutes at Sunday 23:59.

In this project the latter alternative is used i.e. only a single feature is used, representing the minutes elapsed since the start of the week for every product-click event. From now on, this feature will be referred to as $t$. Figure 8 shows a scatter plot of a small dataset where two different playlists are available. The data is plotted with respect to $t$ and the colours of the data points indicate which playlist was shown.
Figure 8: The points represent temporal data of the product-click events and the colour indicates what playlist was shown when the corresponding event occurred.

A better visualization of the dataset would capture the cyclic property of the week. By projecting the points from Figure 8 onto the unit circle, it yields a more natural overview. By defining $t_{tot}$ as the total number minutes in a week (10 080 minutes) and plot $\cos\left(\frac{2\pi}{t_{tot}}t\right)$ on the x-axis together with $\sin\left(\frac{2\pi}{t_{tot}}t\right)$ on the y-axis we obtain Figure 9.

Figure 9: Visualisation of the dataset using the unit circle. Each data point is a product click and the colour indicates which playlist that was used when a product was clicked on. The lines show the start and end of the weekdays.
The lines in the figure shows the start and end of each weekday. In the dataset above there are more product-clicks on Friday than any other day which is easily seen in this view. This method will be used for visualization from now on.

### 3.3 Partitioning the dataset

The idea is to somehow find different subintervals (time intervals) of the week where some playlists seem to produce more clicks than others. To achieve this, the concept of discretization from section 2.3 is applied but is slightly adjusted. To relate to the discussion about discretization, we review the concepts but in this problems’ setting. Let’s call the set with all the available playlists of a video \( S_{pl} \). Then, the examples in the dataset (the product click events) are characterized by the tuple \( e = (t, c) \) where \( t \) is the feature described above and the class value \( c \in S_{pl} \) is an identifier of what playlist was clicked on. The aim is to discretize the range of \( t \) by choosing cut points that partition \( \text{range}(t) \) into subintervals. The cut points are chosen such that the subintervals are characterized by class purity, i.e. the class entropy is low in each of the subsets of examples induced by the subintervals.

Since the feature \( t \) represents the week, which is cyclic, some adjustments from the procedure described in section 2.3, must be done. Fayyad proposed to find a single cut point in each discretization step. When the \( N \) examples are sorted in a sequence there were \( N - 1 \) available cut points at the first sight but the number of possibilities is reduced because of the boundary point property of the minimization of the quantity in (7) concluded earlier in this report. The boundary point minimizing (7) should then be chosen to create two subsets, of the original set of examples, defined by tests on the feature value. The range of the features is thus discretized into a finite set of subintervals.

Because of the search for only one cut point, it is impossible for the example with the lowest feature value and the one with the highest value to end up in the same set (except the case when all points are in the original set). For example, if the point with the lowest \( t \)-value is a product-click early on Monday and the one with the highest value is on late Sunday, these two points could never end up in the same subset after the discretization using the tests in equation (5). This may not be something to be concerned about in cases where a small value and a big value of the feature can indicate that the examples are well separated and it is intuitive that they should not have anything in common.

Even though product-clicks early on a Monday and late on a Sunday will have very different \( t \)-values they are close to each other in time. For example, Sunday and Monday are as close to each other as Monday and Tuesday. Because of that, it is counter intuitive to only find a single cut point with the argument above. Instead, two different cut points are searched for in the first round of the discretization. The cut points are labelled \( T_1, T_2 \) and let \( c(t) \) be the feature value of example \( e \). If the original set of examples is denoted as \( S \), the subsets \( S_1, S_2 \) are then defined by
\[ S_1 = \{ e \in S | \max(T_2, T_1) \geq e(t) \land e(t) \geq \min(T_2, T_1) \} \]
\[ S_2 = S \setminus S_1. \]  

(20)

By using these tests, a partition like the one depicted in Figure 10 is possible to achieve while that would not be possible using the tests in equation (5). The two lines from the origin partition the circle into two segments that represent the achieved subintervals, and the examples in each of these intervals are the obtained subsets. The use of the tests in (5) would fix one of the lines in the figure below to fall on Monday 00:00 while letting the other move freely between the boundary points.

Figure 10: An example of a partition that was impossible to obtain with the tests in equation (5).

This new approach allows for a more flexible choice in the first discretization part. However, the extra degree of freedom does not come for free. If there are \( b \) available boundary points, the number of possible partitions grow as \({b \choose 2}\) instead of depending linearly on \( b \) as in the case when searching for a single cut point.

After the first discretization step is handled separately, the subintervals obtained are further partitioned by searching for a single cut point as described in section 2.3. The procedure continues until the stop criteria are met. The problem of
choosing stopping criteria will be addressed in section 3.5.

3.4 Procedure

When a Smart Video user creates a new video with different playlists, there are no previously collected data. Therefore, all playlists need to be used to evaluate their performance of producing clicks. But how should the playlist be chosen in the beginning? One solution is to introduce randomness. The idea is that the different playlists should be equally probable to be used at first. If there are \( k \) different playlists for a video, labelled \( 1 \ldots k \), the probability that playlist \( i \) should be used in a session should be \( \frac{1}{k} \). In this way, data will be gathered in the database and can later be used in the discretization procedure described in section 3.3.

The Building module is the program that handles the discretization procedure and creates several subintervals of \( \text{range}(t) \). Once all the subintervals are determined, we have obtained subsets \( S_1 \ldots S_n \) that partition the original set \( S \) such that

\[
S = \bigcup_{i=1}^{n} S_i
\]

\[\emptyset = S_i \cap S_j \quad \forall i, j \quad s.t \quad i \neq j.\]

Similarly if \( I_1 \ldots I_n \) are the resulting subintervals of \( \text{range}(t) \) we have that

\[
\text{range}(t) = \bigcup_{i=1}^{n} I_i
\]

\[\emptyset = I_i \cap I_j \quad \forall i, j \quad s.t \quad i \neq j.\]

Each of the subsets carry the examples with a \( t \)-value falling into the corresponding subintervals that are decided by the choice of the cut points during the discretization process. Let \( n_i \) be the number of product clicks produced by the \( i \):th playlist in one subinterval. Then we store the quantity

\[
q_i = \frac{n_i}{\sum_{i=1}^{k} n_i}, \quad i = 1 \ldots k,
\]

for each subinterval. It will be a vector of values that represent the fraction of the product clicks in the subinterval that are coming from each of the \( k \) playlists. The result of the building module will thus be the set of values \( q_i \), one vector per subinterval, together with the corresponding cut points specifying the subintervals that each vector belongs to.

When a page request is incoming and asks which playlist should be rendered the search module will use the quantities that resulted from the building module to deliver the answer. The date and time are transformed to the minutes elapsed since the week start, \( t \), and a lookup procedure starts. The lookup procedure
maps the incoming $t$-value to the subinterval the value belongs to. The statistic $q_i$ will then decide how probable each playlist should be to be used in that session.

The probabilities for each playlist will be uniformly distributed for new videos. When data is gathered the building module will produce different intervals of the week where the different playlists’ probabilities are weighted depending on how many product clicks they have produced in the different intervals before.

The use of randomness is necessary since the module should be rebuilt daily and new statistics need do be available for all playlists. If the classification instead should work by showing the most popular playlist in the interval the module should look the same forever since than certain playlist are always shown in the determined interval. That would not allow for any change in popularity in a videos life time. The case where one playlist is more popular in a specific time interval first but after a while another playlist produces more clicks in the same interval, can be captured by using this method with weighted randomness.

### 3.5 Stopping criteria

In order for the algorithm to terminate, some stopping criteria must be predefined. One interesting quantity for this problem set up is the number of examples of a subset. Since the data points in the subinterval must be used to calculate probabilities, one reasonable requirement would be to stop the partition of a subset if there are few points. Therefore, one stopping criterion is the minimum number of points in a subset, denoted $n_{\text{min}}$. This parameter controls the evolution of the partitioning in the beginning when there are not so many data points, then the building module will just partition the week in fewer intervals. Once new data has been produced the partitioning will lead to more and more subintervals and the resolution will be improved.

Another important parameter is the time span of each of the subintervals i.e. how big the interval is measured in minutes. If the intervals are too small it can result in overfitting, as discussed in section 2.2. Then the week would be partitioned in very fine intervals spanning just a few minutes and perform well on the data at hand. But that would lead to bad generalization of new data. This parameter will be called $t_{\text{min}}$.

At first, the evolution will be controlled by the $n_{\text{min}}$ parameter since there are too few points to form intervals smaller than $t_{\text{min}}$ minutes that contain at least $n_{\text{min}}$ data points. But a video with many product clicks distributed over the week could have more than $n_{\text{min}}$ points in such a small interval and then the parameter $t_{\text{min}}$ will stop these smaller intervals to be formed.

Besides the parameters controlling the size and number of points in a interval, another parameter needed is called $\text{ent}_{\text{min}}$. If a set, that resulted from a previous partition, has an entropy lower than $\text{ent}_{\text{min}}$ it will not be split further and be considered as pure enough. The maximal entropy of a set of $k$ classes is $\log_2(k)$ and this can be used as a reference value when fixing the parameter.
3.6 Algorithms

After the first cut point is chosen according to the description in section 3.3, further partitions of sets are performed by only finding one cut value and using the tests in (5) as described in section 2.3. However, because of the high cost of the more flexible approach in the first discretization step the time complexity of the algorithm will be dominated by the first partition. To be able to run on larger datasets a flag called flexible can be set to true or false depending on if the more flexible first step is suited or not.

The algorithm for finding a single cut point is described in Algorithm 1. The algorithm requires a dataset $S$, a set of boundary points $B$, parameters $n_{min}$, $t_{min}$ to ensure good statistics in the resulting subsets and avoid overfitting.

**Algorithm 1 findCut($S, B, n_{min}, t_{min}$)**

Require:
Finite dataset $S$
Set $B$ of boundary points of $S$
Natural number $n_{min} \geq 1$
Natural number $t_{min} \geq 1$

Begin Algorithm:
1: $T_{best} \leftarrow -1$ \hspace{1cm} \(\triangleright\) Initialize, return -1 if no valid partition was found
2: $e_{min} \leftarrow \text{Ent}(S)$ \hspace{1cm} \(\triangleright\) Calculate Entropy of $S$ according to (6)
3: for every $T \in B$
4: \hspace{1cm} $S_1, S_2 \leftarrow \text{partition}(S, T)$ \hspace{1cm} \(\triangleright\) Split $S$ into $S_1$ and $S_2$ using $b$ as threshold value as in (5)
5: \hspace{1cm} if Valid($S_1, S_2, n_{min}, t_{min}$) then
6: \hspace{1cm} \hspace{1cm} $e \leftarrow \text{E}(T; S)$ \hspace{1cm} \(\triangleright\) Calculate induced weighted entropy according to (7)
7: \hspace{1cm} \hspace{1cm} if $e \leq e_{min}$ then
8: \hspace{1cm} \hspace{1cm} \hspace{1cm} $e_{min} \leftarrow e$
9: \hspace{1cm} \hspace{1cm} $T_{best} \leftarrow T$ \hspace{1cm} \(\triangleright\) Return cut point minimizing (7) and results in valid subsets

For every $T \in B$ the set $S$ is partitioned to subsets $S_1, S_2$ using the function \text{partition}(). The function \text{Valid}() checks if the subsets fulfils the $n_{min}$ and $t_{min}$ criteria. If they do, the quantity (7) is calculated and stored in a variable $e$. The current value of $e$ is stored if it is less than the previously encountered values. The algorithm returns the $T$ minimizing (7) and fulfils the criteria forced on the subsets.

The environment outside the function \text{findCut}() can be seen in algorithm 2. Every resulting subset is placed in a queue four further partitioning. The procedure in the algorithm below is very much like constructing a binary tree. Every node in a binary tree has at most two children. When a set is partitioned, the two resulting subsets can be seen as that nodes children.

25
A leaf is a node without any children, and the subsets that will not be partitioned further is considered as a leaf. Because of the property (21), we only need to store the leaf nodes because all the data is in them. After all leaves are stored, the program iterates over them and calculate the class probabilities (23). Those probabilities are stored in a python list together with the corresponding time interval the leaf represents. The list is sorted with respect to the first cut point in every obtained interval.

Algorithm 2 Discretization procedure

Require:
A queue $q$ holding the resulting subsets from the first partition described in section 3.3
Natural number $n_{\text{min}} \geq 1$
Natural number $t_{\text{min}} \geq 1$
Floating point number $\text{ent}_{\text{min}} \geq 0$

Begin Algorithm:

1: while $q$ is not empty do \(\triangleright\) Repeat as long as there are sets to examine
2: \(S \leftarrow q.\text{dequeue}()\) \(\triangleright\) Pick out a set from queue
3: if $\text{Ent}(S) < \text{ent}_{\text{min}}$ then \(\triangleright\) If the set is pure it is stored as a leaf node in the artificial binary tree
4: \(\text{leaf.insert}(S)\)
5: \text{go to 1}
6: $B \leftarrow \text{extractBoundaries}(S)$ \(\triangleright\) Set of boundary points in $S$
7: $T \leftarrow \text{findCut}(S,B,n_{\text{min}},t_{\text{min}})$ \(\triangleright\) Search for the boundary point minimizing (7)
8: if $T < 0$ then \(\triangleright\) If no valid partitioned was found store $S$ as a leaf
9: \(\text{leaf.insert}(S)\)
10: else
11: \(S_1,S_2 \leftarrow \text{partition}(S,T)\) \(\triangleright\) Partition $S$ into subsets $S_1,S_2$ and insert them in queue for further partitioning
12: $q.\text{enqueue}(S_1,S_2)$
13: for every $S \in \text{leaf}$ do \(\triangleright\) Calculate and store the statistic in leaf nodes
14: \(\text{List.add}((\text{stat}(S))\)
15: \(\text{sort(List)}\) \(\triangleright\) Sort the List w.r.t the first cut point

The date and time of an incoming request are transformed into a $t$-value. A binary search in the list, that resulted from the building module described above, is performed to see which interval the $t$-value belongs to. When the interval is found the probabilities are returned which will be used to choose the playlist. Algorithm 3 describes the procedure.
Algorithm 3 Binary search operation

Require: A list $List$ holding the resulting subintervals with corresponding probability vectors.
A value $v \in \text{range}(t)$ to be searched for.

Begin Algorithm:
1. $L \leftarrow 0$, $R \leftarrow \text{length}(List)-1$
2. while $L \leq R$ do
3. $m \leftarrow \text{floor} \left( \frac{L+R}{2} \right)$
4. $T_1, T_2 \leftarrow \text{interval}(List(m))$ ☰ Extract the cut points defining the interval of the $m$:th element of the list
5. if inInterval($T_1, T_2, v$) then ☰ If the value is inside the current interval, return the probabilities
6. return probabilities($List(m)$)
7. else if $T_1 < v$ then
8. $L \leftarrow m+1$
9. else
10. $R \leftarrow m - 1$
11. return False

3.7 The pipeline

The program is implemented in a Python module called temporalOptimization. Within the module there is a class called Optimize($n_{\text{min}}, t_{\text{min}}, \text{ent}_{\text{min}}$). When creating an instance of the class, the parameters that constitute the stopping criteria are set. A method called build($x$, $y$, flexible) exists within the class. This method is used when the model should be rebuilt. It takes two arrays $x$ and $y$ together with a flag flexible as arguments. The $x$ argument should be an array of $t$ values representing minutes from week start for product clicks. The values in the $y$ array contains integers which indicates which playlist the product clicks belongs to. If there are 3 available playlist the integers in $y$ should be any of 0, 1, 2 etc. The arrays $x$ and $y$ must be of the same length since each value in $x$ should have a corresponding value in $y$. The flexible argument is a boolean and if it is set to true, two cut points will be searched for in the first discretization step allowing for a more flexible result. If it is false only one cut point will be searched for in all of the steps.

By calling the build() method, the decision model will be built and the result will be a python list with intervals and probability vectors as described before. The list is stored in the class member called table. A class method called search($t$) can be called if the table member is not empty. It is supposed to be used when a page request is incoming and the decision of choosing which playlist should be rendered will be made. It takes an argument that is supposed to be a value within range($t$). The method will map the value $t$ to the interval it falls within and return the corresponding probability vector that should later be used to decide what playlist should be shown.
4 Results

4.1 Performance

To test the performance of the program several example sets of different sizes were created. In all tests there were two different available playlists and the \( t \) value were uniformly distributed over the week. There were equally many points for both playlists. The \( t_{min} \) parameter was set to 60 minutes and the restriction of the minimum number of point in a set, \( n_{min} \), was set to 50. Figure 11 shows the running time of the building module depends on the set size. In this experiment the flexible flag was set to \texttt{false} so only a single cut point was searched for in each discretization step. Figure 12 shows the number of boundary points for each set size. Equation (19) should be valid in this case since the points are uniformly distributed and there are equally many product clicks for each playlist. By looking at Figure 12 we see that the equation is justified.

![Figure 11](image)

**Figure 11:** The graph shows how the running time of the building module is affected by the set size. Only a single cut point was searched for in every step.
Figure 12: The graph shows how the number of boundary points depend on the size of the example set.

The time complexity of algorithm 1 is proportional to the product $|B||S|$, where $B$ is the set of boundary points. The reason is that the loop runs over all possible boundary points and the complexity of calculating the entropy is $|S|$. The overall time complexity of algorithm 2 depends on the number of leaves that are allowed to be generated (controlled by the stopping criteria) times the complexity of algorithm 1. In this case the $n_{min}$ was set to 50 and $t_{min}$ to 60 minutes. For small set sizes, the number of leaves generated will depend on the set size because $n_{min}$ dominates as discussed in section 3.5. When there are enough points distributed over the week, intervals smaller than $t_{min}$ will contain enough points so $t_{min}$ will now control the evolution. Since $t_{min}$ does not depend on the set size the number of leaves generated will be constant for large enough sets. Thus the while loop in algorithm 2 will run the same number of times for large enough sets. Thus, the total complexity of the algorithm will be $O(|S||B|)$, which explains the shape of Figure 11.

By repeating the experiment for higher values of $|S|$, the graph seen in Figure 13 is obtained. The curve does not have the same shape as in Figure 11. The behaviour for large $|S|$ can be explained by Figure 14. Since the resolution of the variable $t$ is one minute (see section 3.2) the number of boundary points will approach a constant for large set sizes, namely 10079, which is the maximum value of $t$. The reason is that this number is the number of unique values there are of $t$ and the maximum number of boundary points will be the points between them, as discussed in section 2.3. When the number of boundary points starts to be constant, then the running time will be depend mainly on the calculation of the entropy which is why the graph seems linear for large $|S|$.
Figure 13: The graph shows how the running time of the building module is affected by using even larger set sizes.

Figure 14: The graph shows how the number of boundary points depends on the set size. Because of the finite resolution it will approach a constant for large enough sets.

A similar experiment was carried out but with the `flexible` flag set on `true`. 
Because of high time complexity, $O(|B|^2|S|)$, only set sizes up to a couple of thousands were applied. The result can be seen in Figure 15.

![Figure 15](image)

**Figure 15**: The graph shows how the running time of the building module, when performing a two cut point search in the first step, is affected by the set size.

As observed in the graph above, the runtime increases more rapidly when using the more complex first step.

If the parameter $t_{min} = 60$ the maximum number of intervals that can be produced are $\frac{10080}{60} = 168$. In algorithm 3 the search procedure is shown and since it is a binary search its time complexity is $O(\log(|List|))$. Since there are only around 150 intervals at maximum, the time for a search is approximately zero. The `time` function in python gave 0.000 as result. If the $t_{min}$ parameter was decreased it would result in a larger number of interval, but it would not be a performance problem.

In one element of the resulting list there are two floating point number stored that specifies the cut points and one for each number in the probability vector. If there are three playlists and we store 168 subintervals the total memory required would be around $168 \cdot 5 \cdot 4 = 3360$ bytes.

### 4.2 Tests on artificial datasets

Since only two real world data sets were available, the behaviour of the algorithm is shown in this section by letting it operate on datasets that were created. The parameter values were set to $t_{min} = 60$, $ent_{min} = 0$ $n_{min} = 100$ in all experiments, unless stated otherwise. One data set was created with three
playlists, and all points were completely segregated. This would mean that people only clicked on a specific playlist in different intervals. Of course this would not be seen in real world data sets, but this is just used to show the behaviour. As seen in Figure 16, the algorithm finds the intervals that separates the examples in pure sets. The algorithm stops because there are no boundary points in these subsets.

![Figure 16](image)

**Figure 16:** The behaviour of the algorithm on created data sets where one can separate the examples into three totally pure subsets.

In Figure 17 there are still 3 playlists but the intervals of the points from each playlists overlap in some places. The algorithm finds three pure sets and isolates the parts that are mixed. The mixed intervals are further partitioned since they are big enough and there are enough points within them.
Figure 17: The algorithm finds three pure sets and isolates the mixed regions.

By shifting $ent_{min}$ from zero to 0.45 bits the algorithm does not split a mixed set if the fraction of one playlist is much larger than the other one as seen in the upper part of Figure 18. The lower one shows the result when it is set to zero.
Figure 18: The upper panel shows the result when $\text{ent}_{\text{min}} = 0.45$ whereas the parameter is set to 0 in the lower panel.

4.3 Tests on real datasets

The best way to test the algorithm would be if a real user of SmartVideo would put the program in operation and then evaluate the performance. Unfortunately this was not possible so the tests were performed on one small real dataset and one larger real dataset. The two playlists in each of the data set differed in the
background colour of the product cards, besides that they were the same. In Figure 19 a small real world dataset with 20 product clicks can be seen together with the partition induced by the algorithm. The values of the parameters were $n_{\text{min}} = 4$, $t_{\text{min}} = 60$, $\text{ent}_{\text{min}} = 0$ and 5 subintervals were obtained.

The interval found that ranges from Saturday to Monday only contains product clicks on playlist 2 and is thus totally pure. It may seem strange that the split on Thursday takes place between two points of the same class since that is not a boundary point but the reason is that one of the two adjacent points (marked with the an arrow) are actually two points sharing the same $t$-value but of different classes. In section 2.3 it was stated that the cases of duplicates must be treated as boundary points if there are more than one class present in the group. Since this is a really small dataset the experiment was more intended to show the behaviour of the algorithm rather than draw some conclusions from the statistic in the resulting subinterval (way too few points for that).

![Figure 19](image.png)

**Figure 19:** The resulted partition of a small dataset with two playlists.

The result from a larger real world dataset with 2000 product clicks can be seen in Figure 20. The parameters were set as $n_{\text{min}} = 200$ and $t_{\text{min}} = 60$ minutes. The minimum entropy was set to zero. In the period from Thursday to Saturday the video was more frequently clicked on and one can see in the figure below...
that the partition is finer in that area.

Figure 20: A real world dataset with 20,000 product clicks. The partition is finer in the period Thursday - Saturday because the majority of the clicks occurred in that time period.

The specifications of the intervals and the suggested probabilities in each of them can be seen in Table 1.

Table 1: The resulting intervals and the suggested probabilities to show each playlist.

<table>
<thead>
<tr>
<th>Period</th>
<th>Probabilities (playlist 1, playlist 2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mon 18:34 - Wed 16:00</td>
<td>(0.41, 0.59)</td>
</tr>
<tr>
<td>Wed 16:00 - Thu 15:11</td>
<td>(0.57, 0.43)</td>
</tr>
<tr>
<td>Thu 15:11 - Fri 03:08</td>
<td>(0.43, 0.57)</td>
</tr>
<tr>
<td>Fri 03:08 - Fri 20:05</td>
<td>(0.36, 0.64)</td>
</tr>
<tr>
<td>Fri 20:05 - Sat 12:17</td>
<td>(0.48, 0.52)</td>
</tr>
<tr>
<td>Sat 12:17 - Sun 13:04</td>
<td>(0.38, 0.62)</td>
</tr>
<tr>
<td>Sun 13:04 - Mon 18:34</td>
<td>(0.53, 0.47)</td>
</tr>
</tbody>
</table>

In the table above we can see that the popularities of the different playlist has a bias which is different inside the resulting intervals.
5 Discussion

When the module uses the more flexible first discretization step the result showed that the running time is rapidly growing as the set sizes increase. Therefore, it should only be used on sets with a couple of thousands examples to not be too time consuming. In the case when the flexible choice is disregarded the program performs well even on larger data sets. This is mainly due to the limit of the number of possible boundary points so that the running time only then depends on the entropy calculation time. The IAB recommendation for rendering time limit for personalized ads (300 ms) will not be a problem for this algorithm since the searching module will only look through a small set of intervals and it is a generally simple procedure as the result showed. The scale of memory needed to store the result from the model is below the Mb-scale as required.

To perform a binary search, which is done in this method, the list needs to be sorted which increases the running time of the building module. The advantage is that the time complexity is reduced from be proportional to the number of elements in the list, to proportional to the logarithm of the same number. Since the building module will, at most, be updated a few times a day and the search module will be used very frequently the choice of binary search can be well motivated.

The performance can be improved if the resolution of the $t$-variable is decreased. In this implementation the examples are stored as minutes from week start but by instead measuring the time every ten minutes or quarters will reduce the number of possible boundary points and thus increase the efficiency for set sizes that are strongly dependent on the number of boundary points. By doing that, the flexible first step could be used up to even larger set sizes and produce better results.

Some authors suggests an implementation of a stopping criterion related to the information gain (given by equation (8)) in the discretization process [22]. The motivations are: performing partitions that does not yield any information is unnecessary. If the information gain is not increased, the entropy remains the same. Even though the entropy change is zero, the class probabilities can have been permuted. For example a set with two classes $c_1$ and $c_2$ which corresponding class probabilities $p_1 = \frac{3}{4}$, $p_2 = \frac{1}{4}$ has the same entropy as the set with $p_1 = \frac{1}{4}$, $p_2 = \frac{3}{4}$. In other words, the entropy is not sensible for permutations in probabilities. In cases where purity is the only interest the information gain criterion can be well justified. In this problem a permutation of class probabilities is an important factor since it tells us something about a change in the success of the playlists. By disregarding this criterion, the partitioning will continue until the other stopping criteria are fulfilled and then we have obtained a finer partition where permutation of class probabilities could have been captured.

The algorithm is a heuristic since it chooses the best option in a given step in a top down strategy. It can be shown that the resulting binary tree is not the optimal. However, it is known to be a NP-complete problem to find the optimal decision tree and thus other solutions need to be used [24]. The choice of the
more flexible first partition allows for a better constructed binary tree.

In the only large real data set that was available, the different playlist only differed in background colour of the product cards. It is hard to say how much one would expect that this difference would affect the temporal dependence of the CTR for the two playlists. The creation of the playlists was not intended to capture such a behaviour. However, a bias in the probabilities could be observed in the resulting subintervals.

The behaviour of the algorithm on the created data set was as expected. It found the intervals characterized by purity and the partition was finer at places with more data points and intervals with mixed examples. The best way to evaluating the algorithm would be if a real Smart Video user constructed the different playlist in a way that intuitively could perform better in some periods of the week (day/night or workday/weekend as examples).

5.1 Future work

There are several improvements that can be done to enhance the performance of the implementation. One thing would be to let the user specify the time resolution of the week (which is now set to a minute). This would improve the efficiency in cases where a coarser resolution is acceptable. A more advanced technique for the cut point assignment could also be implemented. In this version the cut points are set as the mid point between two examples. An alternative approach would be to use a biased strategy. When a cut point should be assigned it can be put anywhere between the two examples without changing the resulting subsets. Thus the tree evolution will be exactly the same but a classification of a new example can be different depending on where in interval between the examples the cut point is set. One strategy is to push the cut point towards the smaller set, such that the interval for the larger set is increased and will be more important at the decision making since it has more points and better statistic.

Another improvement would be to adjust the $n_{min}$ criterion. In this version both the resulting subsets of a partition need to carry at least $n_{min}$ examples in order for the split to be accepted. This means that every node in the binary tree that is not a leaf must have always two children. By only requiring one of the subsets to have at least $n_{min}$ examples would increase the flexibility of the algorithm. If $S$ is the original set carrying examples with $t$ value in the interval $I$, and only one subset $S_1$ with examples from interval $I_1 \subset I$ fulfilled the $n_{min}$ criterion the classification needs to be adjusted. An example $e$ would be classified using the examples in $S_1$ if $e(t) \in I_1$, but if $e(t) \in I \setminus I_1$ it would be classified using all the examples in $S$ instead. This procedure would construct a binary tree where the nodes are allowed to only have one child. The search procedure would be more complex if this method was implemented. The reason is that the intervals would overlap and the second relation in (22) would not hold. A suggestion would be to store the resulting interval in a so called interval tree which a nice structure for storing and searching among overlapping intervals [25].

In this method, the whole building process needs to be repeated every time the
model should be updated. One could investigate if it is possible to implement an incremental update scheme so the rebuilding procedure (in its entirety) can be avoided.
References


[20] “Generalized Linear Models” 

[21] “Understanding the Bias-Variance Tradeoff”, Scott Fortmann-Roe


