Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

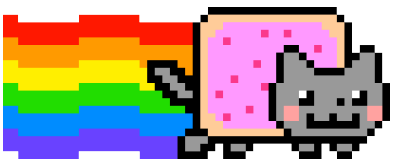
Distributed Computing

Zero-Effort Procrastination

Semester Thesis

Dominik Schlegel
schdomin@ethz.ch

Distributed Computing Group
Computer Engineering and Networks Laboratory
ETH Zürich



Supervisors:
Jochen Seidel, Jara Uitto
Prof. Dr. Roger Wattenhofer

June 23, 2014

Abstract

The main goal of this project is to reduce the amount of unwanted content to a minimum and therefore simultaneously extinguish the efforts for procrastinating. In order to establish such a user-based content selection an active learning algorithm is required.

Our learning algorithm operates on a fixed amount of image data, collected from an online source. In addition to the image data collection we also extract features from the online source. To increase the variety of features even further, we calculate inherent image properties using computer vision algorithms.

We apply the simple, yet powerful, Bayes classification to the derived feature space. This method turns out to be fast enough and sufficiently reliable for our case.

By applying the tuned learning algorithm to a fixed dataset of over 16'000 images and live integration of user-based information, we show that the proposed algorithm is capable of yielding significantly higher user appreciation than a random content selector.

Nomenclature

Notation

$\mathcal{P}(T = A)$	probability of value A of type T to occur
$\mathcal{P}(T = A\|U = B)$	probability of value A (type T) to occur, given the occurrence of value B (type U)
$\mathcal{P}(T = A\|U = B, V = C)$	probability of value A (type T) to occur, given the occurrence of value B (type U) and C (type V)

Acronyms and Abbreviations

ZEP	Zero-Effort Procrastination GUI Application
Meme	Digital image displaying a specific idea, style or action (in the context of this project)
Memebase	Web Page hosting memes, http://memebase.cheezburger.com/
MySQL	Open Source Database, http://www.mysql.com/
OpenCV	Open Source Computer Vision, http://opencv.org/
Git	Git Code Management, http://git-scm.com/
GIF	Graphics Interchange Format, http://www.w3.org/Graphics/GIF/spec-gif89a.txt
JP(E)G	Joint Photographic (Experts) Group Graphics Format, http://www.jpeg.org/
GUI	Graphical User Interface
IDE	Integrated/Interactive Development Environment
HTML	Hypertext Markup Language, http://www.w3.org/html/
URL	Uniform Resource Locator, http://tools.ietf.org/html/rfc1738
phpMyAdmin	MySQL Browser, http://www.phpmyadmin.net/

Contents

Chapter 1

Introduction

1.1 Motivation

Procrastination is an inseparable, important part of our daily life and yet hardly valued appropriately. Once embraced by the tempting grasp of procrastination we lose track of time and crucial open issues get decelerated or abandoned. This seems to be a rather negative outcome but one should not forget about the relaxing and regenerative effects resulting from procrastinating. Besides profiting from those beneficial elements one might even solve open issues or make them obsolete simply by putting the mind in a new scenario.

Unfortunately successful procrastination is not in all cases effortlessly possible. As an example, directly relating to our cause, one could visit a web page in expectation to gain some relief by looking at funny pictures but is still forced to observe unwanted images while browsing.

We want to create a GUI (Graphical User Interface) application displaying predominantly appreciated images. This way the effort required for procrastination can be minimized. We will elaborate on this "application" in the coming sections.

1.2 Related Work

The probably most prominent example relating to this project is Googles' advertising component Google AdSense [1]. Similar to ZEP the objective is to autonomously display content based on user information. In this case the content is an advert from a web page in a web browser. Naturally Google collects enormous amounts of user data and is therefore absolutely capable of predicting a users taste very precisely. Their application performs very well and many of us have already been or will be astonished by the amazing accuracy of some advertisements while browsing. An alike behavior can also be observed on the pages of online stores suggesting additional items which may be of interest to the user (e.g. "Customers Who Bought This Item Also Bought" on Amazon [2]).

1.3 Target Application

In this section we define the desired final functionality of our application and list the corresponding requirements for the implementation. We distinguish between *application* and *algorithm*. The term application is used to describe the complete, final software piece whereas algorithm is used to indicate individual procedures running inside the application. We will also refer to the application as ZEP.

1.3.1 Basic Idea

The final application should provide a reactive GUI, displaying mostly liked images upon user requests. In order to determine how much an image will be liked we integrate an active learning algorithm in our application. The learning algorithm is called active because it evaluates live user input and continuously adapts its classification parameters to the new input. We use the term *classification* to describe the computation and assignment of a specific value (class, e.g like-probability) to an image, indicating how much it will be liked. This specific value is also referred to as *classification value*. We make use of the term *learning* to outline consecutive, adapting classification phases and the dynamical procedures required by them.

The algorithm knows the total number of images available. To successfully classify an image, the algorithm requires a distinct set of properties corresponding to the image, so called *features* (e.g. text amount, tags). The features should exist for every image and may take on different values. We use the term *feature type* to describe only a certain kind of features (e.g. image size).

The set of images utilized by ZEP is autonomously obtained from an online source. Depending on the layout of the online source we can directly extract features from the web elements surrounding an image. We compute additional features by applying computer vision algorithms to the images. The online feature extraction and the custom feature computation processes are further summarized under the term *feature generation*.

The *dataset* is the collection of all images and their corresponding features. After importing the dataset, the learning algorithm is able to classify every image based on its features. As soon as we computed the classification values for all images, we rank them and display the highest ranked one to the user.

In order to improve our prediction accuracy and actually *learn* over time, we continuously should integrate the user input on displayed images. The user input is nothing but a Like or Dislike label. The process of assigning a Like or a Dislike label to an image is called *labelling*.

1.3.2 Implementation Requirements

In order to provide the desired functionality for our target application we have to implement the following core components:

- **Data Acquisition**: Autonomous gathering and online storage of the image information of the internet, including feature generation.
- **Data Analysis**: Evaluation and selection of the dataset and feature types followed by the computation and online storage of additional static properties required by the learning algorithm.
- **Data Import**: ZEP should import as much data as possible at start-up to minimize delays during the interactive phase.
- **Active Learning**: Based on the precomputed feature information for each image and the real-time, continuous user input on images (labelling), the task of ZEP is to classify a set of subsequent images and order them by their classification values.
- **User Interaction (GUI)**: Only the title and the corresponding image are shown to the user. The user has 3 possibilities to interact with the GUI:
  - Like: Labels the current image as liked and notifies the learner.
  - Dislike: Labels the current image as disliked and notifies the learner.
  - Previous: Displays the previous image, thus enabling relabelling of any image already visited.

  After the user labelled an image ZEP should display the next, most likely to be liked image based on its classification value.

Note that the term *data* is used to describe a bigger range of information (images, features, learning parameters) than dataset while including the dataset as well.

1.3.3 Programming Language

The programming language set for this project is Java. We briefly introduce the major external libraries, being language features, used in our project:

- **OpenCV** (v2.4.8): The prominent open source computer vision library OpenCV [3] is used to calculate features from images.
- **mysql-connector-java** (v5.1.29): This library allows easy and fast interfacing with a MySQL database [4].
- **jsoup** (v1.7.3): Jsoup is used for HTML parsing and Javascript querying.

We choose Eclipse [5] as IDE and Git for revision control.

1.4 Challenges

The main challenges to expect are briefly formulated below.

Feature generation is of major importance since it directly affects the learning process. The challenge lies in picking and creating the most suitable feature types for our classification target.

The final performance is resulting solely from the **real-time classification** of ZEP. Besides choosing the best learning algorithm the application should run smoothly and not block at any time during the user interaction phase. Multi-threading and optimized database operations are considered.

1.5 Limitations

There exist several areas with limitations worth considering:

- **Image content**: All images are retrieved autonomously from a web page without further filtering. The content is therefore directly defined by the community of that web page.
- **Feature information**: Most of the feature types are plainly extracted from the web context of the images. This bounds the learning capabilities to the quality and quantity of community inputs (title text, tags) to an posting.
- **User information**: The received user information during the active learning phase is not guaranteed to be relatively rational and independent. The performance evaluation is therefore not 100% reproducible.
- **Prediction accuracy**: Since the complete user input consists only of a single label per image the prediction accuracy is also restricted. In contrast to the possibilities we might have resulting from scanning the users' entire internet browsing history.

Chapter 2

Approach

2.1 Learning

In this section we elaborate on the learning algorithm. We introduce the fundamental concept behind the algorithm, list the features we use for the classification and thoroughly explain the final learning procedure.

The learning process is the core component of ZEP since its provided classification results are crucial for the ranking of images and the subsequent selection of images to display.

2.1.1 Concept

The appropriate learning method is selected based on the desired classification output. In our case the classification should provide enough variety to order the images by the outcome. Therefore we intuitively prefer a learning algorithm which does not just classify the images as likeable or dislikeable, but assigns the probability of being one of the two classes to the images.

A publicly available algorithm, perfectly fulfilling these requirements, is the Naive Bayes classifier [6]. This algorithm is built upon Bayes' theorem [7].

Theorem 2.1. *Bayes' theorem*

$$\mathcal{P}(T = A|U = B) = \frac{\mathcal{P}(U = B|T = A)\mathcal{P}(T = A)}{\mathcal{P}(U = B)} \tag{2.1}$$

We demonstrate the method of the Naive Bayes classifier using our classification requirements on a simple setup with 2 feature types.

Note that with this method we are only able to evaluate features by being present or not (that's what the probability values stand for in our case). This limitation is more extensively discussed in the coming passage about feature types (Section 2.1.2) and will be more clear after the following example.

We first introduce the following event types with respective values:

Label Y:	$Y \in \{1 : \text{Like}, 0 : \text{Dislike}\}$
Feature (of type 1) F_1:	$F_1 \in \{1 : \text{Present}, 0 : \text{Not Present}\}$
Feature (of type 2) F_2:	$F_2 \in \{1 : \text{Present}, 0 : \text{Not Present}\}$

The probability we are interested in is:

$$\mathcal{P}(Y = 1 | F_1, F_2 = \{1, 0\})$$

This value describes the chance of obtaining a Like, depending on the presences of the features F_1 and F_2. The problem with this property is that we cannot directly compute it. Thanks to Bayes theorem (Theorem 2.1) we are able to rewrite the expression as:

$$\mathcal{P}(Y = 1 | F_1, F_2 = \{1, 0\}) = \frac{\mathcal{P}(F_1, F_2 = \{1, 0\} | Y = 1)\mathcal{P}(Y = 1)}{\mathcal{P}(F_1, F_2 = \{1, 0\})} \tag{2.2}$$

with the terms:

$$\mathcal{P}(F_1, F_2 = \{1, 0\} | Y = 1) : \text{Conditional expression to resolve (see below)} \tag{2.3}$$
$$\mathcal{P}(Y = 1) : \text{Prob. of a Like (identical for all images)} \tag{2.4}$$
$$\mathcal{P}(F_1, F_2 = \{1, 0\}) : \text{Prob. of occurrences for } F_1 \text{ and } F_2 \text{ (indep.)} \tag{2.5}$$

In order to resolve Equation 2.3 we apply the rules for conditional independence. This is possible since the features F_1 and F_2 are both conditionally independent on the Label Y. We obtain:

$$\mathcal{P}(F_1, F_2 = \{1, 0\} | Y = 1) = \mathcal{P}(F_1 = \{1, 0\} | Y = 1)\mathcal{P}(F_2 = \{1, 0\} | Y = 1)$$

These separate, conditional probabilities of a Like for particular feature types (present or not) can be computed.

The probability of a Like (Equation 2.4) is irrelevant to us since we are finally only interested in the ranking of the images. The multiplication of all images with the same value does not change the ranking sequence.

By applying the above findings and use of the law of independence on Expression 2.5 we can simplify Equation 2.2 to:

$$\mathcal{P}(Y = 1 | F_1, F_2 = \{1, 0\}) = \frac{\mathcal{P}(F_1 = \{1, 0\} | Y = 1)\mathcal{P}(F_2 = \{1, 0\} | Y = 1)\mathcal{P}(Y = 1)}{\mathcal{P}(F_1 = \{1, 0\})\mathcal{P}(F_2 = \{1, 0\})} \tag{2.6}$$

This equation defines the classification value of an image. The formula can easily be extended to work with an arbitrary number of feature types, as long as they are conditionally independent on the Label Y.

2.1.2 Images and Features

The next important step is the discretization of the features we want to use for this classification. The resulting feature types define the different probabilities required by our formula. As a consequence, the choice of suitable features is crucial for the performance of the learning algorithm.

In our case, the most intuitive way to obtain features is by directly extracting them from the web context of an image. The source of our images is also the source of our features.

In order to select the most appropriate data source for our project, we should get an overview over the different layouts of web pages hosting images. Our most relevant findings, followed by the choice of the final image and feature source are discussed in the next passage:

Source Selection

A referencing system, only sometimes seen on web sites hosting images, is the individual marking of uploaded content with text attributes, so called *tags*. The tags are set by the person publishing the image on the web page and can vary in length and number. Besides tags there exist also more common properties such as image title, upload date, Like/Dislike counters and comments history.

Because of their individuality and the direct relation to an image, tags are predestined to be some of the most significant features. We therefore focus on the tag topology of a page while looking for a suitable source.

Mostly because of the superior tag quality but also the appealing image content we select the web page:

`http://memebase.cheezburger.com/`

as our image and feature data source. The site provides access to the history of almost 5'000 pages, containing over 20'000 images.

Having the feature source and therefore the layout of the web page (HTML) we can define the feature types to extract. The feature types have to be of the same present/not present nature as required by our classification Formula 2.6. This is no issue for the tags, but for other features such as Like/Dislike counters we have to come up with a sufficiently representative and meaningful present/not present feature type. Next, we elaborate on the feature types, including the representation formulas.

Feature Types

The feature types derived from our data source, including the values they can attain, are introduced below:

- Feature type **Tag**: The tags form the most important feature type. This type is particularly different from the other feature types because it can occur more than one time per image (e.g. an image with 4 tags contains 4 features of type tag and only 1 feature of type title). Logically, an image has to provide always at least one tag to be useful for our algorithm. The tag feature type can obtain the following values:

$$T = \begin{cases} 1 & \text{if the tag is present (character string matches)} \\ 0 & \text{else} \end{cases}$$

  This means the tag *bananas* is present for an image containing the tag *bananas*, but not for an image with the tag *banana*. If we only care about the information whether an image contains at least one *banana* or none, one of the tags is redundant. Redundant tags reduce the number of images per tag, also referred to as *tag density*, by increasing the amount of tags.

  The condition is very simple and at the same time crucial for our learning performance. It might be possible that a high tag density, which is desirable for the learning algorithm, is difficult to achieve depending on the tagging quality of the online community.

- Feature type **Liked**: This feature type is a combination of the Like/Dislike counters (n_{Likes}, $n_{Dislikes}$):

$$L = \begin{cases} 1 & \text{if } \dfrac{n_{Likes}}{n_{Dislikes}+1} > 5 \\ 0 & \text{else} \end{cases}$$

  The idea behind this feature type is to stress images that are highly appreciated by the community. The cutoff value (5 in this case) can be adjusted to raise or lower the required Likes/Dislikes ratio for a *Liked* image.

- Feature type **Hot**: The *Hot* feature type is a more experimental combination of the Like/Dislike counters and the number of comments ($n_{Comments}$):

$$H = \begin{cases} 1 & \text{if } \dfrac{n_{Likes}+n_{Dislikes}}{n_{Comments}+1} < 10 \\ 0 & \text{else} \end{cases}$$

  As for the *Liked* feature type the principle is to emphasize certain images depending on the community responses.

- Feature type **Animated**: This feature type simply separates animated images (GIF) from regular ones (JPG, JPEG):

$$A = \begin{cases} 1 & \text{if the image type is GIF} \\ 0 & \text{else} \end{cases}$$

  Due to the display problems of GIF images, the learning algorithm is currently not considering animated images. This feature type is therefore obsolete until the GIF functionality of ZEP is re-enabled.

To further supply our learning algorithm with feature information we introduce sophisticated image properties computed by the use of computer vision algorithms (OpenCV [3]). The feature types are defined as follows:

- Feature type **Text**: The amount of text in an image is a relevant attribute since some people might prefer images without much text on them while others enjoy reading very detailed comic strips. The problem with this attribute is that the web page naturally does not care about this information and hence we have to compute it by ourselves. The computation is explained in the image analysis paragraph (Section 2.3.2). As the amount of text in an image is known we can define our feature type:

$$X = \begin{cases} 1 & \text{if text amount} > 0.25 \\ 0 & \text{else} \end{cases}$$

  This feature type basically tells the learning algorithm whether an image contains a relatively high amount of text or not.

- Feature type **Photograph**: Another attractive property is the actual kind of an image. If we are capable of distinguishing between (computer) drawings and real photographs we can spare a user from getting flooded with drawings, while he is only interested in photographs. As for the *Text* feature type we have to compute this property by ourselves. The photograph feature type takes on the following values:

$$P = \begin{cases} 1 & \text{if the image is a } \textit{photo} \\ 0 & \text{else} \end{cases}$$

  The determination of a *photo* is thoroughly described in the image analysis paragraph (Section 2.3.2)

Next, all the feature types mentioned above are summarized in a table and the resulting final classification formula, based on those feature types, is introduced.

Feature Type	Letter	Derivation	Parameter Value	States
Tag	T	Presence	none	$\{0, 1\}$
Liked	L	Cutoff	5	$\{0, 1\}$
Hot	H	Cutoff	10	$\{0, 1\}$
Animated	A	Boolean	none	$\{0, 1\}$
Text	X	Cutoff	0.25	$\{0, 1\}$
Photograph	P	Empirical	various	$\{0, 1\}$

Table 2.1: Features table, containing all feature types.

Image Classification Formula

Knowing all feature types (T, L, H, A, X and P) from Table 2.1, we can redefine Equation 2.6 into our final classification formula:

$$\mathcal{P}(Y = 1|\sum_{i}^{N}[T_i], L, H, A, X, P) = \frac{\prod_i^N [\mathcal{P}(T_i|Y)]\,\mathcal{P}(L|Y)\mathcal{P}(H|Y)\mathcal{P}(A|Y)\mathcal{P}(X|Y)\mathcal{P}(P|Y)\mathcal{P}(Y=1)}{\prod_i^N [\mathcal{P}(T_i)]\,\mathcal{P}(L)\mathcal{P}(H)\mathcal{P}(A)\mathcal{P}(X)\mathcal{P}(P)} \tag{2.7}$$

Where N is the total number of tags for the current image and T_i is tag i of the current image. The respective values of the random variables have been excluded to improve readability. For Equation 2.7 the following formulations are equivalent:

$$\text{for } F \in \{T, L, H, A, X, P\}: \qquad \begin{aligned} \mathcal{P}(F|Y) &\Rightarrow \mathcal{P}(F = \{1, 0\}|Y = 1) \\ \mathcal{P}(F) &\Rightarrow \mathcal{P}(F = \{1, 0\}) \end{aligned}$$

Even though the formula and its components are deceptively easy to understand, the actual computation and application of it by the learning algorithm seems to be quite abstract. In the following section we address this issue by introducing the complete learning procedure, including the realization of the classification formula.

2.1.3 Algorithm

For the description of the learning algorithm we take the following elements for granted:

- Image Provision: The learning algorithm is provided with the complete image database and can arbitrarily select images from it.
- Feature Provision: The learning algorithm contains the complete feature information, including the number of occurrences of each feature type in the database.
- User Input: The resulting label (Like/Dislike) and the corresponding image after a user action are directly routed to the learning algorithm.

Layout

We define the structure of the learning algorithm as follows. Afterwards, we will elaborate on the components of this layout.

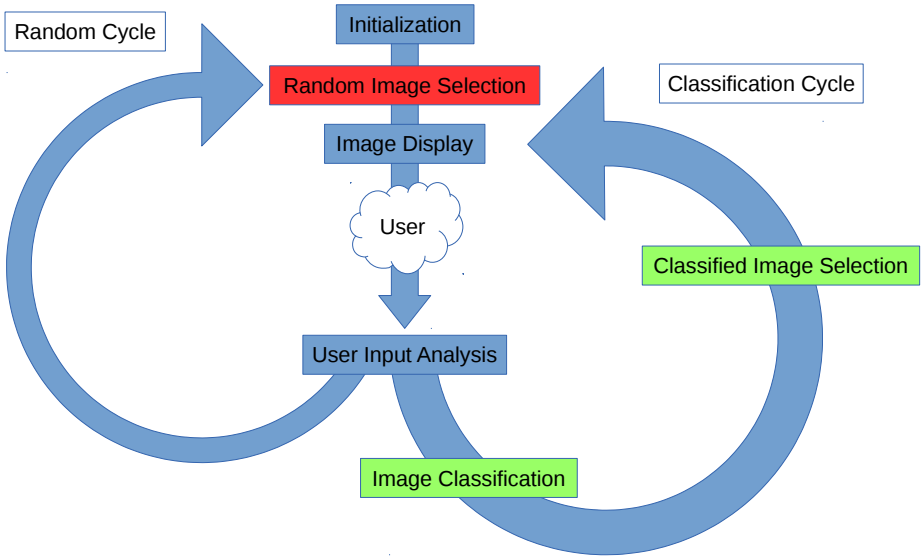


Figure 2.1: Visualization of the learning algorithm. Either one of the loops is executed every time the user demands a new image. The algorithm keeps running until the user closes the application.

Initialization

The first thing to do for our algorithm is to pre-compute all constant values to avoid unnecessary overhead during the user interaction phase. By taking a closer look at our Classification Formula 2.7 we find that the probabilities in the denominator are absolute and therefore remain constant. The probabilities can be directly computed from the number of occurrences of feature types in our database:

For any feature type F:

$$\mathcal{P}(F=1) = \frac{n_{(F=1)}}{n_{(F=1)} + n_{(F=0)}} = \frac{n_{(F=1)}}{N_F}$$
$$\mathcal{P}(F=0) = 1 - \mathcal{P}(F=1)$$

Where $n_{(F=1)}$ is the number of occurrences of this feature type having value 1 (being present) and n_F is the total number of occurrences of this feature type in our database.

During the Initialization phase we compute the potential denominator of our Classification Formula 2.7 for all feature constellations. The only part still missing is the numerator, formed by the independent, conditional probabilities. We apply a very simple method to compute the conditional probabilities:

For any feature type F:

$$\mathcal{P}(F=1|Y=1) = \frac{n_{(L,F=1)}}{N_F}$$
$$\mathcal{P}(F=0|Y=1) = \frac{n_{(L,F=0)}}{N_F}$$

Where N_F is the total number of occurrences of the current feature type in our database and $n_{(L,F=1)}/n_{(L,F=0)}$ are the number of likes of images containing/not containing the feature type F so far.

This naive approach offers several advantages but also some disadvantages. Major advantages are its simplicity and sufficient functionality regarding our task. A downside lies in its naivete, assuming that a liked feature type automatically implies the users preference for this feature type, which might not be true.

It is obvious that this method requires a history of liked images to work. Hence the learning algorithm has always to start with a random image calibration phase before starting to suggest images.

The only thing we can do during the initialization phase regarding the conditional probabilities, is setting the liked counters for all feature type values to zero. The counters get incremented after the evaluation of the user input during the User Input Analysis.

Random Image Selection

At this point we know the probabilities of all features for being present in an image. But we do not have any (first image) or require random user information (subsequent images) to compute the conditional probabilities.

A random image is picked uniformly from the remaining images in the database. Since random images are potentially unwanted by the user their number has to be kept as small as possible and Random Image Selection should be avoided whenever feasible.

Image Display

The image display starts when the learner sends a fresh image (classified or random) to the GUI and ends if the user labelled the image and requests a new one. Between those time points the learning algorithm remains idle.

User Input Analysis

After the user labelled an image as Liked or Disliked it is transferred to the idle learning algorithm. If the image was liked the learning algorithm increases the corresponding liked feature type counters. If the image was disliked the learning algorithm only notices the dislike and ignores the feature information. This section appears to be critical because we have an intentional loss of information. We chose this solution to avoid the loss of probabilities if the user suddenly starts to dislike images. In order to fix this issue we would have had to come up with an alternative learning method.

After the User Input Analysis the learning algorithm decides if a classification based on the current like counters is meaningful or if a random image should be displayed. The decisioning is currently manually set by a fixed number of repetitions per cycle.

Image Classification

In the classification phase the Classification Formula 2.7 is applied to all remaining images in the database. The probability function (see Section 2.1.1) for a Like of any image is defined as follows (for completeness):

$$\mathcal{P}(Y = 1) = 0.5 + \frac{n_{Likes} - n_{Dislikes}}{2N}$$

Where n_{Likes} respective $n_{Dislikes}$ is the current number of liked/disliked images and N is the total number of images available in the database.

Classified Image Selection

The Classified Image Selection is the counterpart to the Random Image Selection. All remaining images get ranked by their classification value in descending order. The top image(s) are then transferred to the GUI for the Image Display. Optimally the learning algorithm always uses the classified path to display images. Only in that way the effort can be minimized.

In order to increase the application speed, always a bundle of the best, classified images (currently 10 images) is selected instead of just one. Then the Image Classification can be run simultaneously in an additional thread, while the user is clicking through the bundle of previously selected images.

2.2 Data Acquisition

This section is providing an overview of the data acquisition process required by our application. As stated by the implementation requirements (Section 1.3.2), the extraction and storage of the data should be as autonomous as possible. The process of going through online content and extracting information is also referred to as *Web Crawling*. We use the term Web Crawler to label the part of our application responsible for the data acquisition.

We distinguish between image data (actual images) and feature data (fields such as title, Likes counters, Dislikes counters, etc.). As data source for both types serves the online Memebase: `http://memebase.cheezburger.com/`

2.2.1 Image Data

In order to access online image data for manipulation, we have to find the hosted location of an image by its URL. The easily readable HTML structure of the Memebase web page allows a simple detection of the body containing the storage URL for a particular image. We can retrieve the image URL from the HTML body with a simple HTML parser. Once the image location is known we can download the image or upload a copy of it to our database.

2.2.2 Feature Data

The HTML parsing for the feature data is not as simple as for the image data. Since the amount of tags per image is not fixed we have to parse the content generically. Most features are easy to obtain since they consist only of a single fixed field entry in the HTML code. To have persistent access to our extracted data we should insert it into a suitable database structure. The same database is used for the two types of data.

2.3 Software Design

Figure 2.2 shows an overview of our software components and their interactions:

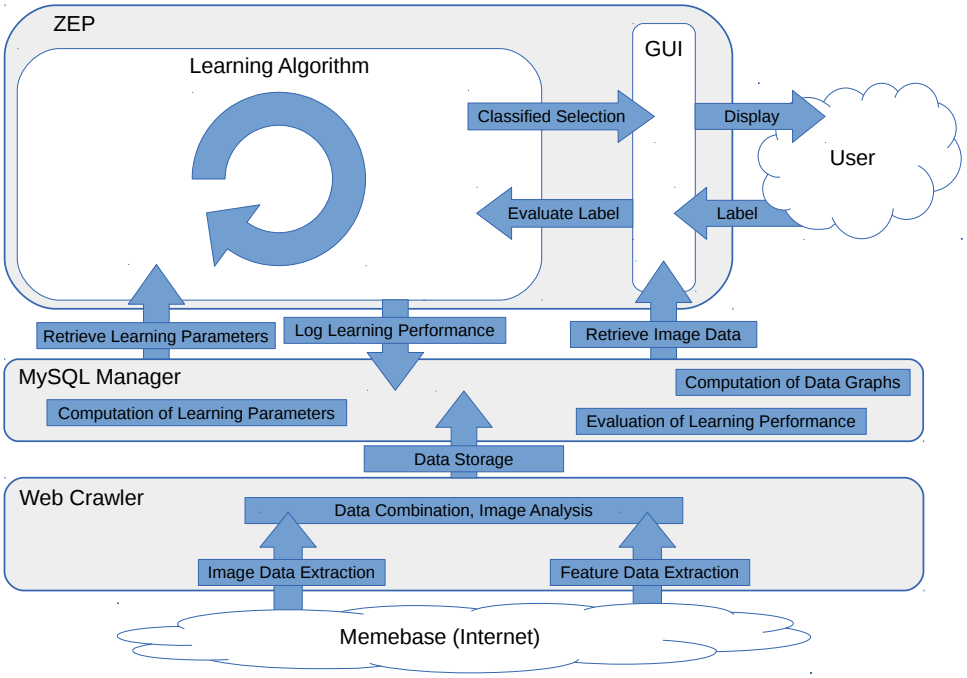


Figure 2.2: Chart of the software scheme. The structure is defined by the 3 main components Web Crawler, MySQL Manager, and ZEP. All major interactions between these components are marked with arrows, where the arrows' orientation indicates the direction of the communication between them.

The details of the learning algorithm are described in Section 2.1.3. In some cases the element box is not part of an arrow (e.g. Computation of Learning Parameters), which indicates that this functionality is triggered manually if required. Since the MySQL Manager manages our database, most evaluative functions are also implemented there.
In the following passages we provide implementation details of the major components including relevant sub units.

2.3.1 Web Crawler

The Web Crawler is a independent application inside the project. Its purpose is the autonomous acquisition and transformation of image and feature data from the Memebase web page including final storage in the MySQL database.

The Web Crawlers main functionality lies in HTML parsing. As discussed in the Data Acquisition passage (Section 2.2) almost all required data has to be obtained from HTML text. The additional feature generation by the use of computer vision algorithms is elaborated in a separate section due to its complexity. Nevertheless this post-feature-generation, or so-called *Image Analysis* is part of the Web Crawler.

This condition leaves us with the pure HTML parsing for this paragraph. We used the Java component Jsoup to parse the HTML document into an tree structure.

With the Jsoup attr function most of the required fields could easily be accessed. The Likes and Dislikes counters get set by Javascript and are not included in the initial HTML document. We managed to access the internal URL, used to retrieve the counters of a particular image by Javascript, over the Firefox Web Debugger. All acquired data is finally stored in our MySQL database.

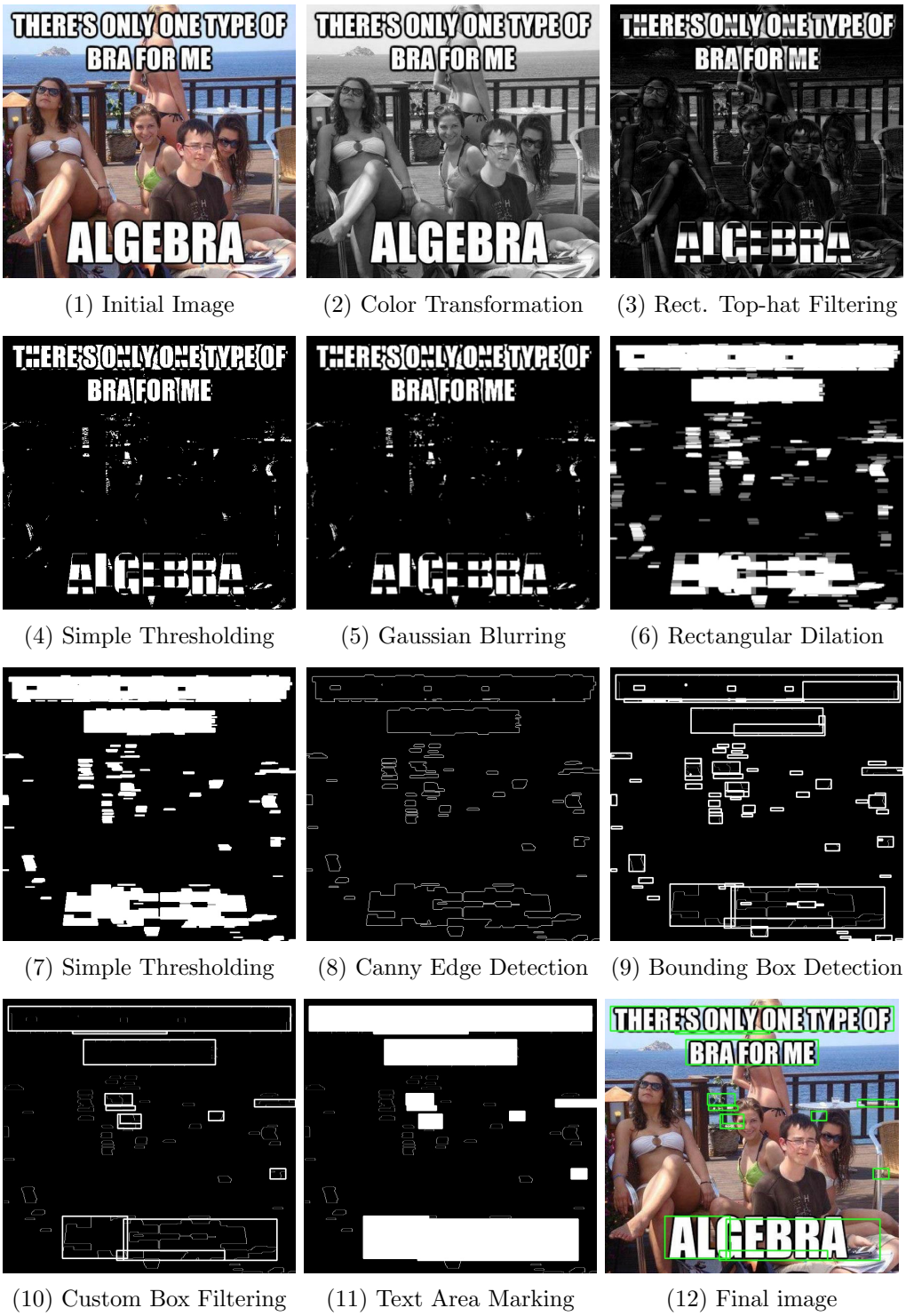
2.3.2 Image Analysis

For the derivation of the 2 additional feature types Text and Photograph (see Section 2.1.2) we apply OpenCV [3] algorithms to the image data.

Text Detection

We start by computing the amount of text in an image. Text detection is a very challenging task when aiming for high precision and reliability. Our case does not require high precision but merely a reasonably low false positives rate to provide a useful feature.

Upon receiving an image a series of OpenCV functions is executed on the image raw data. After the successful execution of the 9 steps we receive an image containing only a few large but many small boxes (Figure 2.3.9). Most of the small boxes are caused by noise in the image and do not actually indicate a text area. It is also possible that boxes appear completely within another box. In order to get rid of the noisy small boxes and the double boxes, we apply several filtering functions during the post-processing phase (Figure 2.3.10). The final result of a text detection case and snapshots of the passed stages during the process are shown in Figure 2.3.



(1) Initial Image (2) Color Transformation (3) Rect. Top-hat Filtering

(4) Simple Thresholding (5) Gaussian Blurring (6) Rectangular Dilation

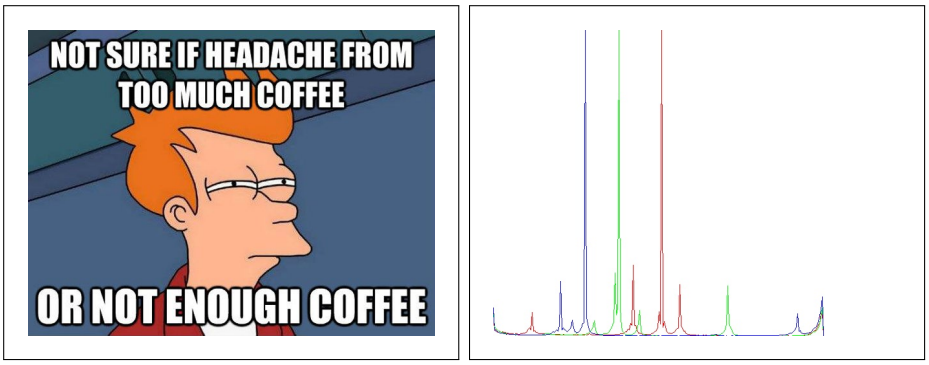
(7) Simple Thresholding (8) Canny Edge Detection (9) Bounding Box Detection

(10) Custom Box Filtering (11) Text Area Marking (12) Final image

Figure 2.3: Compilation of image RGB contents after the respective text detection stages. The final detected text areas are marked by the green frames in the last image (2.3.12).

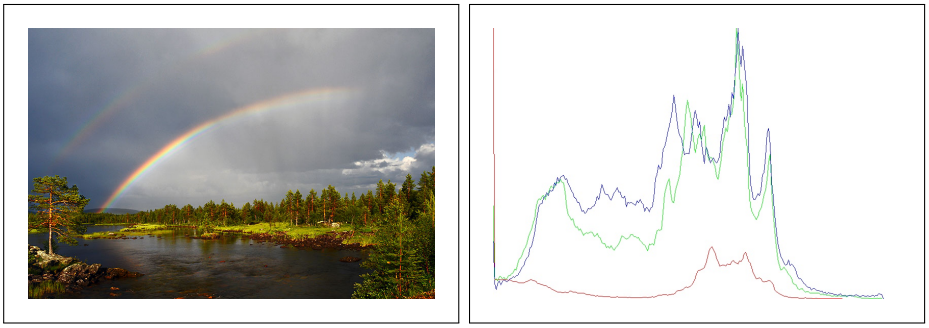
Photograph Detection

For the second computer vision feature type, we separate drawings from photographs. A simple approach is to evaluate the color histogram for an image and define the minimum variety of colors required for being a photograph. From the histogram we compute the maximum averaged value, the standard deviation and the mean. By measuring the variations of those attributes for different images we derive certain thresholds to determine if an image is a photograph or not. Below we discuss two images with typical RGB histograms:



(1) Drawing (2) RGB Histogram Drawing

Figure 2.4: Photograph Detection: Drawing. The RGB histogram of the drawing consists of almost only peaks for each color type. This indicates that large amounts of the respective color tone have been used in the image (e.g. the blue color of the wall behind Fry [8]). The distinctive spiky shapes appear for most histograms of drawings.



(1) Photograph (2) RGB Histogram Photograph

Figure 2.5: Photograph Detection: Photograph. In the RGB histogram of the photograph we can observe a much smoother distribution of the color values over the image.

Since the high color variety is guaranteed for most photographs we are able to distinguish photographs from drawings just by looking at the histogram plots in Figure 2.4 respectively Figure 2.5. Our implementation compares the shapes of histograms by the attributes of the respective curve. In addition to the RGB histogram also a simple Grayscale histogram can be used with only minor losses in prediction accuracy, enabling a significant speedup.

2.3.3 MySQL Manager

The MySQL database [4] of our project was hosted by the ETHZ network. The database can be managed manually over the phpMyAdmin browser interface. A direct interface to our database is implemented by the CMySQLManager class.

Most of the required I/O functionality is provided by the java.sql package. The link to the database is established using the JDBC connector. All MySQL selections and insertions are done using the prepared statement technique.

In order to use the MySQL functionality a component has to instantiate the MySQL manager with the correct configuration parameters. Parallel MySQL operations are supported and exception handling immediately reports errors during the execution of crucial procedures. The MySQL Manager is a standalone component and can therefore be run independently on its own.

In addition to its MySQL interface the manager also provides the following triggerable functionality (see Figure 2.2):

- Computation of Learning Parameters: Once the feature and image data is acquired, we can compute static properties such as the feature type probabilities.
- Computation of Diagrams: The complete data acquisition is logged in the MySQL database. Hence we can easily compute graphical representations of specific properties in our database (Section 3.1).
- Evaluation of Learning Performance: The learning algorithm continuously logs its current learning parameters to the MySQL database. For the final performance evaluation we only have to assess the information in our database. This process is implemented by this function.

2.3.4 Learning

We realized the method introduced in Section 2.1.3 in the CLearnerBayes class. The feature type probabilities are organized through HashMaps. As many steps as possible are implemented as individually as described in Section 2.1.3.

In addition to the learning functionality the learner provides the GUI with fresh images. The learner does not store the actual image data but only the index of the image in the database. The largest data structures inside the learning implementation are the Maps holding the feature information.

For the performance evaluation we log every image selection and the corresponding user response to the MySQL database. As previously discussed (Section 2.3.3), the evaluation of the learning sessions is done by the MySQL Manager. The learner itself has no means of checking its live learning performance.

2.3.5 GUI

Another crucial component of our project is the GUI. The most important requirements are discussed in Section 1.3.2. We implement the CGUI class using the javax.swing package. The component is directly instantiated by the Java main method and notifies the learning algorithm for image requests or labels. Additionally, we integrate the KeyListener interface to enable convenient keyboard control of the GUI buttons. A Screenshot of the final GUI layout of ZEP is displayed in Figure 2.6:

Figure 2.6: Screenshot ZEP GUI. The interactive elements Previous, Dislike and Like are clearly visible and mark their controllability by the arrow keys. The window is scalable and freshly requested images automatically adapt their size to the new frame size.

Chapter 3

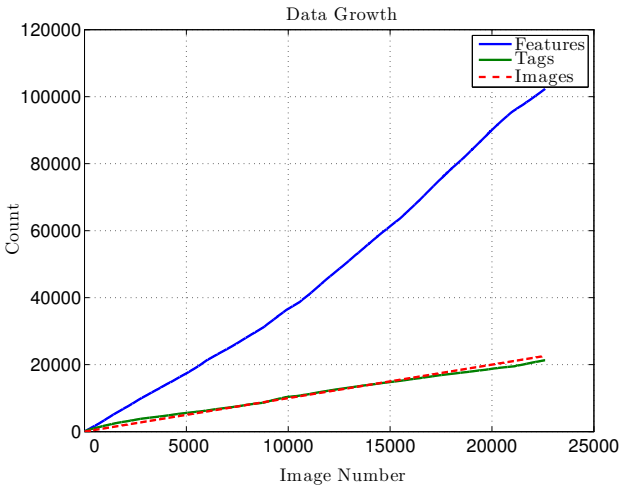
Results

3.1 Data Acquisition

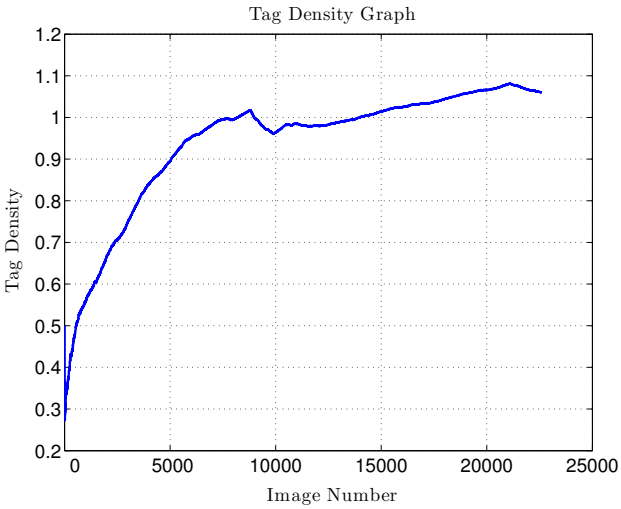
In this section we show multiple diagrams deduced from the autonomous Web Crawling phase (defined in Section 2.2).

3.1.1 Data Growth

The Crawling was done within two sessions. The numbers of images from both Crawling sessions add up to over 20'000. Both sessions were executed on a single Laptop and took over 24 hours in total to complete.



(1) Images and Features Growth



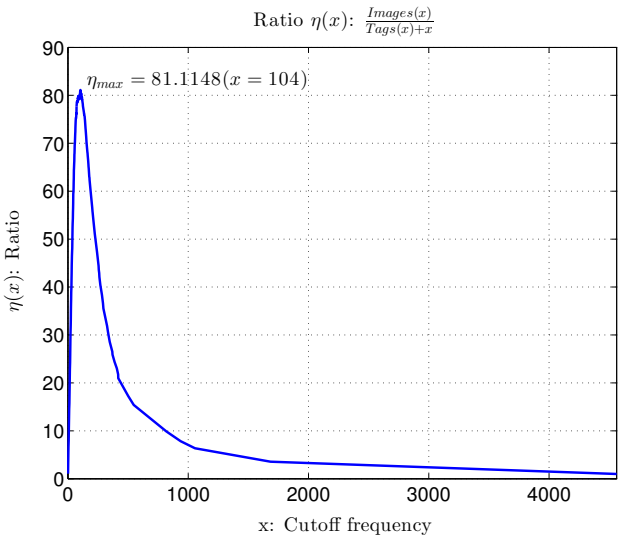
(2) Tag density

Figure 3.1: Comparison of the images and features topology. The growth of the tags is also displayed separately. The initially positive trend and the final, unsatisfactory result of the tag density (defined as the ratio: $\frac{Images}{Tags}$, see Section 2.1.2) can be observed in the diagram to the Right.

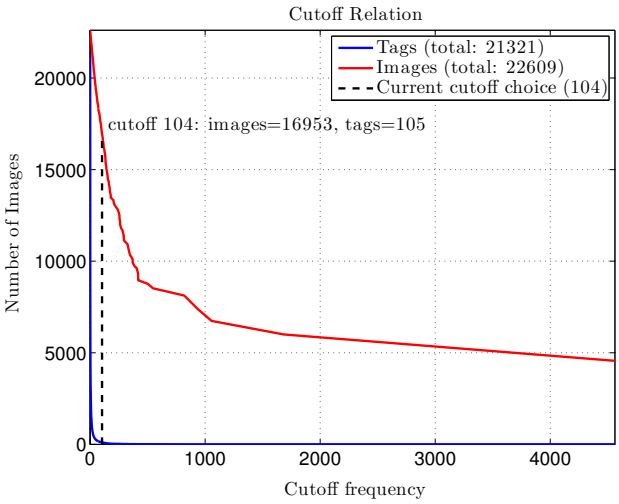
3.1.2 Tag Frequency Cutoff

In order to raise the tag density and therefore strengthen the Tag feature type we chose to filter our database by tag frequencies. By *tag frequency* we understand the number of appearances of a certain tag in different images. The filter pass criterion was the minimum tag frequency at least one tag of an image had to provide.

As a result, images containing only nearly unique tags got discarded and the final tag density could be raised by a factor of over a hundred compared to the original situation. This was only possible due to the negligible number of discarded images relative to the total dataset. The effects of the minimum tag frequency (also referred to as *tag frequency cutoff*) on the dataset are shown in Figure 3.2, including our final choice for the cutoff value.



(1) Utility Function: Ratio $\eta(x)$



(2) Relation: Images to cutoff

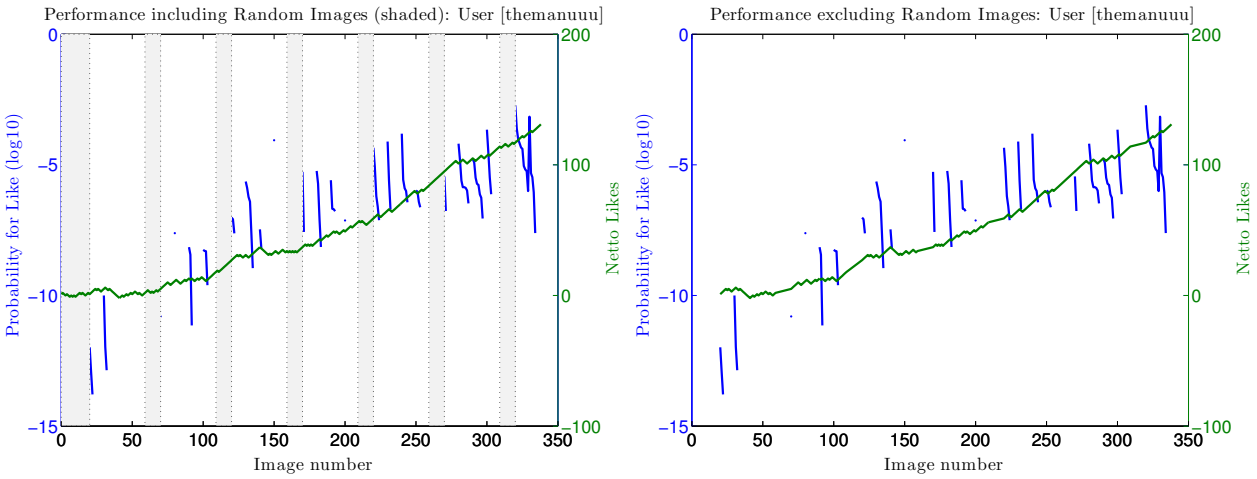
Figure 3.2: Display of the tag frequency cutoff evaluation. The decision for the best cutoff was made by defining a utility function, expressing the normalized tag frequency, and computing the position of its maximum value (shown in the left figure, Figure 3.2.1). To the Right (Figure 3.2.2), we drew the number of images for a particular cutoff. The cutoff value of our choice is marked by the dashed, vertical line.

This choice defines the dataset for the learning algorithm. With a cutoff value of 104 we possess 16'953 images to display.

3.2 Learning

The point of this section is to clarify the assessment of the performance data from the learning algorithm. We will discuss the results firstly in the next chapter. We cover the performance evaluation of 3 distinct sessions to mediate our findings. The evaluations can be computed any time after the user closed the application. All performance evaluations of ZEP were performed unsupervised. We provided people interested in testing with the required binaries including simple scripts to run the application.

We start with a performance diagram of ZEP in one of the first configurations (Figure 3.3). The configuration is determined by the settings of random and classification cycles and the images available to classify. We defined the properties *Netto Likes* ($n_{Likes} - n_{Dislikes}$) and *Probability of Like* (classification value of an image) as the most relevant elements regarding the performance diagram. By analysis of the correlation between the two resulting curves one can directly draw conclusions about the influence of ZEP. In the first configuration ZEP had only access to a limited amount of images for classification (*Limited Classification*).

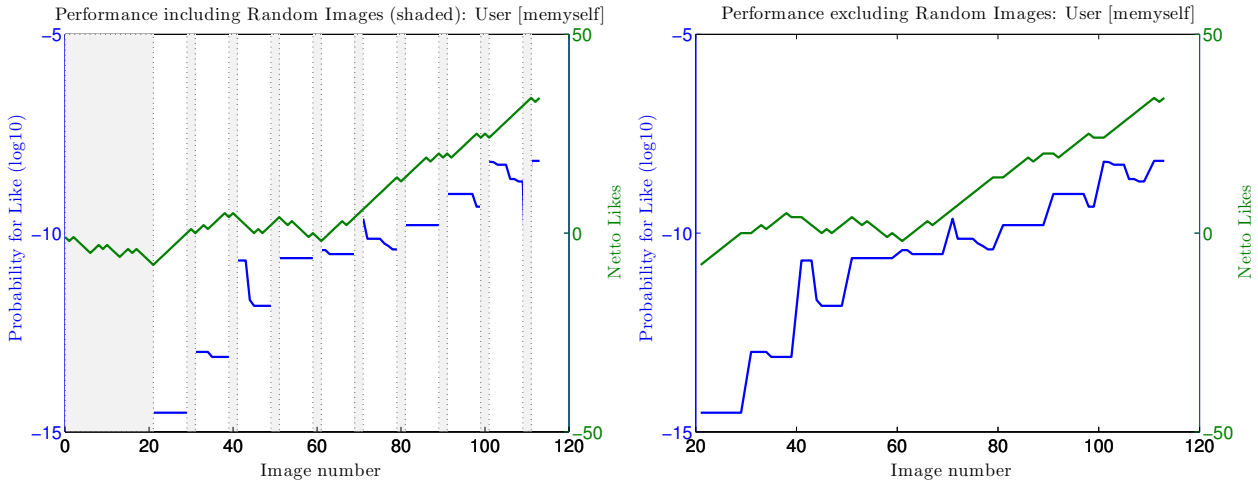


(1) Performance inlcuding Random Selection

(2) Performance without Random Selection

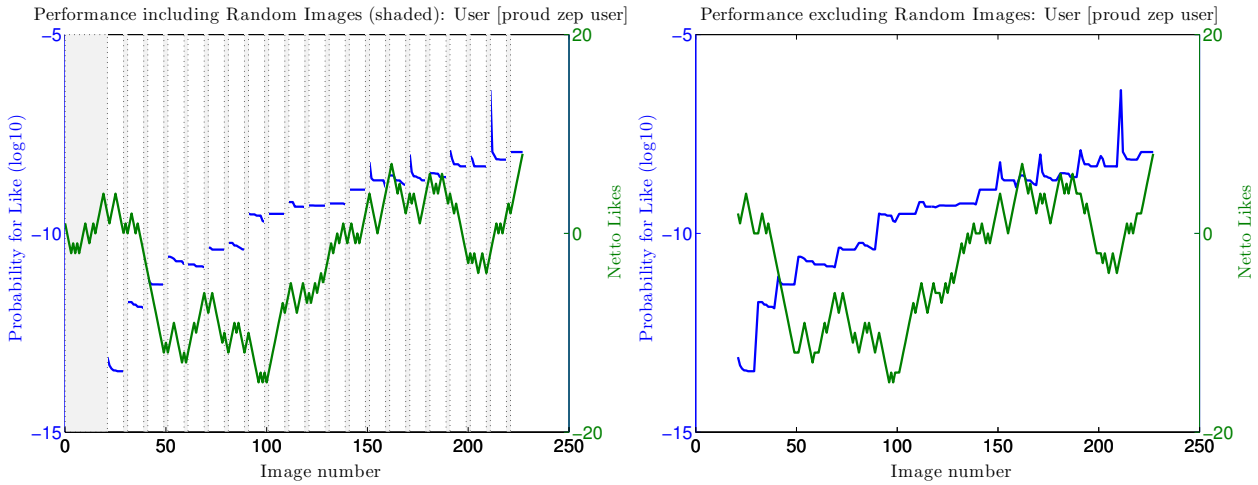
Figure 3.3: Learning Performance with Limited Classification. For this case we set the Initial Random Selection cycle to 20 images and every 50 subsequent images we triggered another Random Selection cycle for 10 images. The random cycles are the shaded bars visible in the left figure (Figure 3.3.1).
The probability curves (blue) and the reason for their thread-like appearance in this case are discussed in the next chapter. The right figure (Figure 3.3.2) excludes the Random Selection cycles and shows the pure influence of ZEP.

Figure 3.4 and Figure 3.5 show the performance of ZEP with the final configuration. For both evaluations the initial Random Selection phase lasts 20 cycles and we insert only one randomly selected image after every series of 9 classified images. Additionally we allowed the learning algorithm to access the complete dataset for each classification (*Unlimited Classification*).



(1) Performance inlcuding Random Selection (2) Performance without Random Selection

Figure 3.4: Learning Performance 1 with Unlimited Classification. The probability curves appear smooth and the correlation of the two curves is easier to observe than in the previous situation (Figure 3.3).



(1) Performance inlcuding Random Selection (2) Performance without Random Selection

Figure 3.5: Learning Performance 2 with Unlimited Classification. This sample is included for the result analysis because of its inconsistent, flickering Like rate.

Chapter 4

Discussion

4.1 Achieved Performance

In this passage we evaluate the correlation between the curves in the diagrams (see Section 3.2) to draw conclusions about the performance of ZEP.

4.1.1 Limited Classification

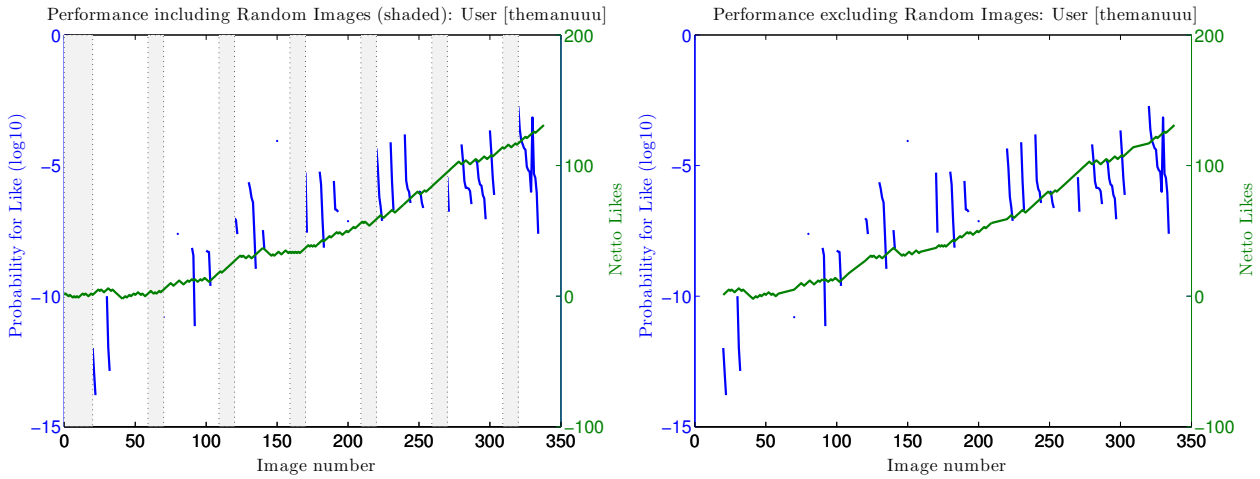


Figure 4.1: Learning Performance with Limited Selection. The thread-like probability strips (blue) in Figure 4.1 occur because the learning algorithm runs out of matching features to classify the images, due to the limited image pool available for classification. This means that ZEP could not provide Classified images for selection during the Classification cycle and therefore the real amount of random images is higher than indicated.

The beneficial effect of ZEP on the Netto Likes curve is easily observable, but since the Classification cycle was limited, the relation is not completely transparent.

4.1.2 Unlimited Classification

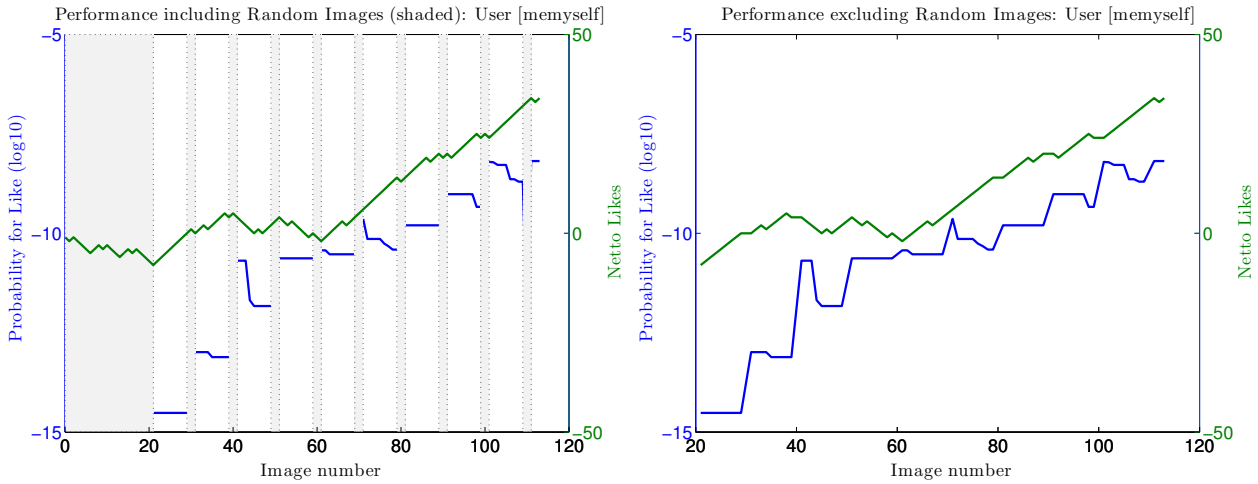


Figure 4.2: Learning Performance 1 with Unlimited Classification. The calibration phase is completed after roughly 60 images and the nicely correlated curves underline the accurate predictions of ZEP.

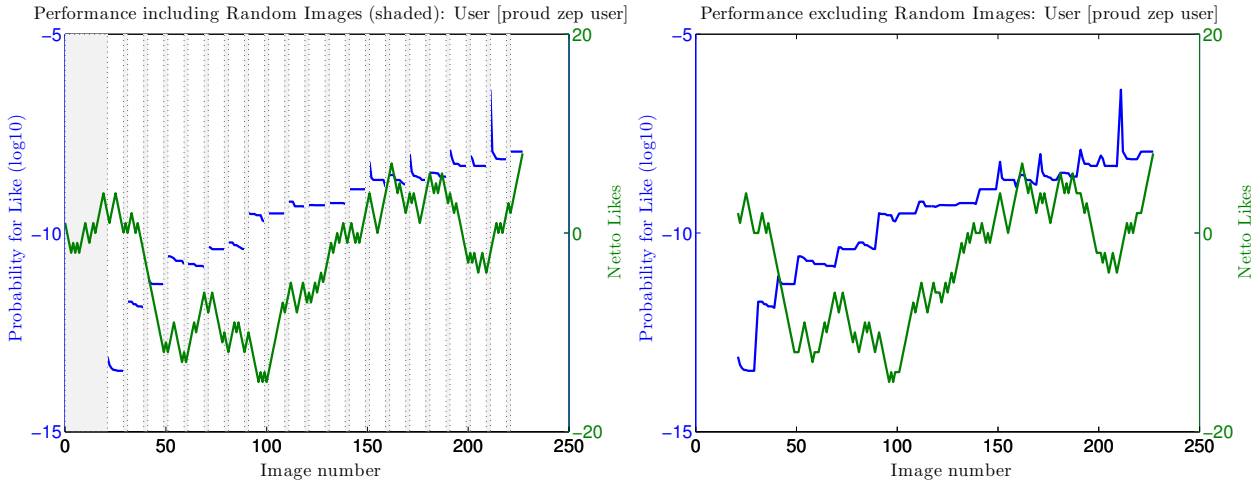


Figure 4.3: Learning Performance 2 with Unlimited Classification.

We analyse Figure 4.2 and Figure 4.3 simultaneously. Both diagrams were obtained with the Unlimited Classification configuration. There are no more thread-like probability function values visible as in Figure 4.1 because of the unlimited amount of images (from the dataset) to classify. For Figure 4.3 we observe a completely different behavior for the first 60 images. The rapid growth of Probability for a Like goes hand in hand with a rapid decrease of the Netto Likes indicating that the user strongly disliked the content presented by ZEP.

This distinct phenomena (the same situation occurs again at around 100 images in Figure 4.3) is a result of the Unlimited Classification configuration. Because of the availability of all images the learning algorithm is capable of providing images which perfectly match the liked content so far. This means that if images with highly overlapping features are liked, the learner will provide similar ones in his turn. Optimally this leads to even more Likes but it can also annoy the user since the content variety shrinks dramatically, which happened in this case. We call this effect *spiral* because of its increasingly growing effect. Besides the negative spirals we can observe positive predictions for a slight majority of images.

4.1.3 Tag Analysis

After the performance evaluation sessions we computed rankings for the top 10 liked and disliked tags. Figure 4.4 shows the resulting diagrams.

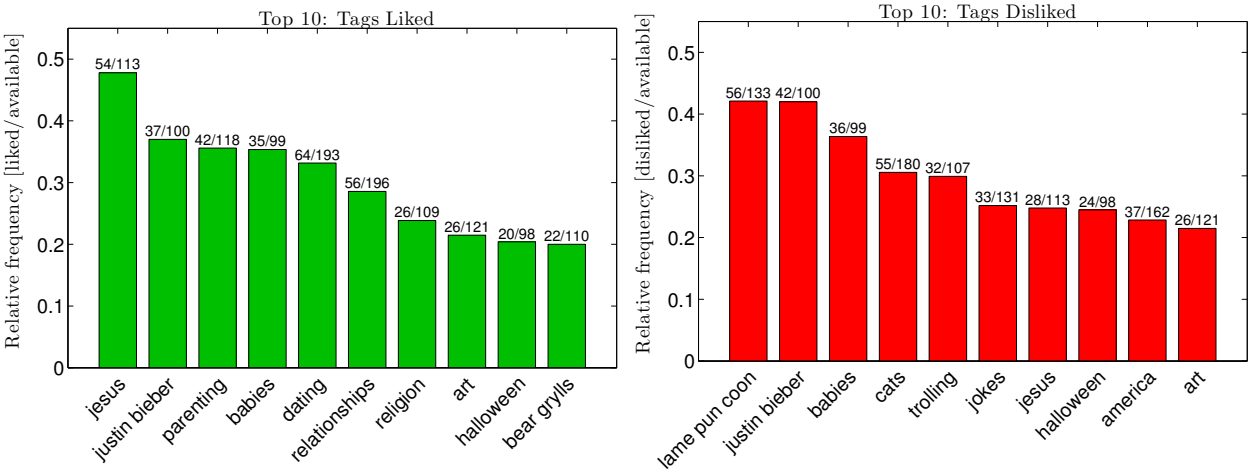


Figure 4.4: Top 10 Tags over all Performance Evaluations. In order to avoid the ranking of only tags with high frequencies we normalized the tag counts by their respective frequency. The numbers at the top of the bars display this method. Bars with a high number of tag frequencies indicate a strong presence in describing the overall users taste.

Note that some tags show up in the Liked as well as in the Disliked ranking (e.g. justing bieber, halloween and even jesus). This might be because of the previously mentioned spiral effect (see Section 4.1.2). From that we can tell that a user started to like predominately images with the same tags (e.g. justin bieber) and got caught in a spiral. The user naturally started disliking the incoming, annoying flood of images containing mostly these tags and this is the reason why we obtain the same tags in both rankings in the end. On the other hand, these double listings could also result simply from opposing user tastes.

4.2 Final Conclusions

Conclusively we can say that ZEP achieved its target of reducing the amount of effort for procrastination. For most of the users ZEP was able to recommend a majority of appreciated images. But there were also cases where ZEP was not able to sustainably provide images the user liked. Some possible solutions for this matter are discussed in Section 4.4.

4.3 Applications

The purpose of ZEP is to enhance procrastination. ZEP is attractive for people regularly browsing funny picture web pages (such as Memebase). This group of people is usually highly content sensitive and each individual has its own, preferred source of images. Since ZEP is currently bound to the Memebase it is difficult to be a considerably tempting alternative to the users favorite web page.

For an alternative application, the manually triggerable performance evaluations of ZEP could allow web pages to benchmark their images. In this way, they could establish a high quality of their image database.

4.4 Future Work

There exist various enhancements applicable to ZEP. We list a selection of what we consider the most important:

- User Profile: In the current state, all learning parameters for a specific user get discarded as the ZEP applicationg gets closed. By enabling a persistent login for a user and storage of all learning sessions, ZEP could create a user specific profile. The profile could be loaded as soon as the corresponding person logs into ZEP and the person could enjoy effortless procrastination from the start.
- GUI Modification (GIF): Animated images (GIF) are currently disabled because of display issues. If those issues can be resolved we could also extend our image selection to animated images and provide a whole new content spectrum.
- Web Page Selection: All images currently displayed originate from the Memebase (`http://memebase.cheezburger.com/`). By enabling the crawling of custom image sources we could provide images from the preferred source of the user without him bothering to view unwanted images. In this state ZEP, could become a real threat to browser based image viewing.

Appendix A

Application Guide

A.1 Dependencies

The complete source code is available on: `pc-5413.ethz.ch/domis.git`

The ZEP main method is located at:
`domis/src/main/CMain.java`

The Web Crawler main method is located at:
`domis/src/crawling/CCheezcrawler.java`

The MySQL Manager main method is located at:
`domis/src/utility/CMySQLManager.java`

The project can be imported into Eclipse with the following steps:

1. File ⇒ New ⇒ Java Project
2. Location: ZEP folder (src directory is not visible)
3. Project Name: domis (same as the ZEP folder after checkout)
4. Default Options ⇒ Finish

In order to compile and run the Web Crawler, the OpenCV library has to be set up for Java Eclipse: `http://docs.opencv.org/2.4.4-beta/doc/tutorials/introduction/desktop_java/java_dev_intro.html`

Additionally a runnable version of ZEP is provided in a zip file.

A.2 Tutorial

All information required to run ZEP is included in the readme file:

```
1  ------------------------------------------------------------
2  README: ZEP Debug Build XXXX-XX-XX
3
4
5
6  ------------------------------------------------------------
7  RUN GUI:
8
9  - open a console
10 - change to directory: zep
11 - make sure the folders bin and thirdparty are visible
12
13 LINUX:
14 - run:
15 java -cp bin/.:thirdparty/*: main.CMain
16
17 WINDOWS
18 - run:
19 java -cp bin\.;thirdparty\*; main.CMain
20
21
22
23 ------------------------------------------------------------
24 KEY BINDINGS:
25
26 [Escape]         : Quits GUI and closes the application
27 [LEFT ARROW KEY] : Visit previous image - deletes the last classification action
28 [DOWN ARROW KEY] : Classifies the current image as Dislike - opens next image
29 [RIGHT ARROW KEY]: Classifies the current image as Like - opens next image
30
31
32
33 ------------------------------------------------------------
34 TROUBLESHOOTING:
35
36 - Command java not found: make sure the java binary path is set in the $PATH environment variable
37 - Make sure to use a unique username without any magic, hebrew letters
38 - Continuous logging available through the console
39
40
41
42 ------------------------------------------------------------
43 CONFIG FILE:
44
45 - Create a local file: config.txt in the zep directory (top level)
46 - Make sure to set the MySQL login information correctly (username, password)
47 - The file has the following syntax (only the lines with = signs matter):
48
49 //ds GUI
50 m_iWindowWidth=1200
51 m_iWindowHeight=800
52
53 //ds MySQL
54 m_strMySQLServerURL=jdbc:mysql://pc-10129.ethz.ch:3306/domis
55 m_strMySQLUsername=username
56 m_strMySQLPassword=password
```

Bibliography

[1] Google: Inside AdSense. http://adsense.blogspot.ch/ (Accessed: 10. June 2014).

[2] Greg Linden, B.S., York, J.: Amazon.com Recommendations. Industry Report (January/February 2003)

[3] Developers, O.: Open Source Computer Vision. http://opencv.willowgarage.com/ (Accessed: 10. June 2014).

[4] Oracle: MySQL: The world's most popular open source database. http://www.mysql.com/ (Accessed: 16. June 2014).

[5] Foundation, E.: Eclipse Java IDE. http://www.eclipse.org/ (Accessed: 12. June 2014).

[6] Caruana, R., Niculescu-mizil, A.: An empirical comparison of supervised learning algorithms. In: In Proc. 23 rd Intl. Conf. Machine learning (ICML 06. (2006) 161–168

[7] Hazewinkel, M.: Bayes formula. Encyclopedia of Mathematics (2001)

[8] Greenwald, S.J.: Klein's beer: Futurama Comedy and Writers in the Classroom. Primus: Problems, resources and issues in mathematics undergraduate studies. Volume 17 (2007) 52–66