# Extracting orthogonal binary codes from images

Author: Michele Ricciardi

Supervisor: Dr Novi Quadrianto

*University of Sussex, Department of Informatics*

Master of Science in Intelligent Systems

2015

# Acknowledgements

First and foremost I would like to thank my supervisor, Dr Novi Quadrianto, for his continued support during the project and for introducing me to such an interesting field of research, which was previously unknown to me.

I'd also like to express my gratitude my director and colleagues for supporting me and providing enough flexibility to allow me to proceed and complete this dissertation project.

Last but not least I would like to thank my partner, family and friends for providing moral and emotional support.

# Abstract

Binary codes are useful representations of images used in image retrieval and classification. Similarly to other techniques that compress large matrices into smaller representations, there is a fine balance between discarding as much information as possible whilst retaining the predictability of the data.

This project focuses on extending an existing method of generating binary codes by adding an orthogonality constraint between each of the binary codes. The idea behind this approach is to create features that have a visual meaning to humans and that encompass different aspect of an image.

This report covers the theory behind the orthogonality constraint and how it was added to the original method. As well as covering the theory, the practical details of the implementation are covered explaining how the optimization problem was solved and implemented.

An extensive comparison between the original method and the modified algorithm is covered in this report, comparing and contrast discriminative results as well as displaying the results in some visualization.

# Table of Contents

# Table of Figures

# 1. Introduction

Thanks to more powerful mobile cameras and the ever growing number of social media websites, in recent years the amount of pictures taken everyday has increased exponentially.

On Instagram alone, 70 millions pictures are shared every day averagely. (Instagram - Press, 2015)

Thanks to this ever growing number of visual sources of information both the research community and corporations have taken great interest in extracting information from images and finding better ways of indexing, searching and classifying images.

One specific area of research is binary representations of images. Binary representations are interesting because they can be used for many machine learning tasks such as classification and retrieval with great performance over other representations.

The aim of this project was initially to build a machine learning application using an established method of extracting binary codes from images. However after finding some shortcomings of the original method, the project steered towards trying to overcome those shortcomings and therefore trying to improve the original method.

The work conducted during this project is summarized in this report, highlighting what the main issue was with the original method and creating a new problem formulation to try and overcome the issue. Additionally the details of the implementation of the revised method are highlighted, explaining which parts of the original code were modified.

After describing the changes to the problem formulation and its implementation, the report describes the evaluation method that was put in place in order to compare the performance of the original method and the revised method.

Lastly the results of the experiments are reported, compared and discussed.

# 2. Background

“There has been increasing interest in building search/index structures for performing similarity search over high-dimensional data, e.g., image databases, document collections, time-series databases, and genome databases” (Gionis, Indyk, & Motwani, 1999) and those searches often suffer from the “curse of dimensionality”. For this reason various techniques have been explored to try to reduce large dimensional spaces in binary codes.

“Binary codes are attractive image representations for image search and retrieval, because they are easy to match, and the capacity of the space of very short binary codes is so large that all of the digital images in the world can be indexed with relatively short binary codes.” (Rastegari, Farhadi, & Forsyth, 2012)

However, generating binary codes is no easy task.

Initially the problem was seen simply as a dimensionality reduction problem, hence techniques like PCA where firstly explored to reduce the dimensionality of the data. However when reducing dimensionality using PCA, we are essentially creating a mapping between the initial dimensions and the reduced dimensions. This means that although we get smaller dimensions, the resulting features are not really “simple” and not binary.

So in 2001 a new idea was explored which to create very simple features representation, which the authors described as “machine learning approach for visual object detection which is capable of processing images extremely rapidly and achieving high detection rates” (Viola & Jones, 2001) In their method they used Haar-like box functions to classify images of faces, and their method of simplifying the features used, gave some very good results in that context.

Since then binary representations have really taken off, and in 2011 Gong & Lazebnik summarized how an effective method of producing binary codes should be built:

“An effective scheme for learning binary codes should have several properties. First, the codes should be short so that we could store large amount of images in memory. [..] Second, the codes should map images that are similar (either perceptually or semantically) to binary strings with a low Hamming distance. Finally, the algorithms for learning the parameters of the binary code and for encoding a new test image should be very efficient. The need to simultaneously satisfy all three constraints makes the binary code learning problem quite challenging.” (Gong & Lazebnik, 2011)

Gong & Lazebnik describe a method called Iterative Quantization (also known as ITQ).

A number of methods have been explored over the years, with some of the most notable ones using hashing functions. With the main variants being locality sensitive hashing (Gionis, Indyk, & Motwani, 1999), parameter sensitive hashing (Shakhnarovich, Viola, & Darrell, 2003), kernelized locality sensitive hashing (Kulis & Grauman, Kernelized locality-sensitive hashing for scalable image search., 2009), binary reconstructive embedding (Kulis & Darrell, Learning to hash with binary reconstructive embeddings., 2009) and semantic hashing (Salakhutdinov & Hinton, 2009)

In the paper “Attribute Discovery via Predictable Discriminative Binary codes” by Rastegari et al (2012), the authors claim to “differ from these constructions, because our method is explicitly discriminative. Furthermore, instead of learning bits independently, we learn bit vectors as a whole.” (Rastegari, Farhadi, & Forsyth, 2012). As it will be described later in this report, in the method described by

Rastegari et al the learning problem is explicitly focusing on making each of the binary codes discriminative. They do this by minimizing the distance of similar images and maximizing the distance between dissimilar images (more details will be covered in later sections of this report).

In this paper the authors make an in depth comparison between state of the art methods and their solution and they demonstrate that their method performs better than state of the art methods.

As this method outperforms other state of the art methods, it was decided to proceed with this method to build the 20 questions game.

# 3. Application

In the first part of this project the option of creating an application which utilized the binary codes was explored. The idea was to build a "20 questions game" using the binary codes generated using the method described in Rastegari et al (2012).

The concept of the 20 question game is that the application asks the user to think of an object or person and after asking 20 questions the application guesses what the user was thinking of.

Binary codes would be the perfect image representation for this type of application.

The main idea is that the user would have to reply Yes or No to each question and by doing so the application can start forming the binary code based on each response (Yes=1, No=0). So At the end of the 20 questions the application will have a 20-bit long code which can then be used to search through the dataset and understand which object/person the user was thinking of.

The initial thought was that once the binary code was formed (i.e. when the user answered all of the questions), the application would do a kNN classification using the Hamming distance as a metric.

The initial thought was that this could be achieved using the binary features extracted using the method described in Rastegari et al (2012). This is because each of the binary code created is an "unnamed" visual characteristic of a given image and whilst an application cannot name the feature itself, a human should be able to easily recognize such visual characteristic.

Before pursuing the idea of building an application using the binary codes few considerations were made.

Firstly it was necessary to find an appropriate dataset for the task. As animals are quite easily recognizable by humans and some of them can be recognized by unique features such as stripes, spots, fur etc. a dataset of animals would be an idea for such game. A great animals' images dataset was found called Animals with Attributes (Lampert, Nickisch, Harmeling, & Weidmann, 2008-2015). This dataset

contains 30475 images of 50 animals and it comes with different extracted features.
The dataset chosen for this potential application was the DECAF features dataset.

Secondly once the dataset was chosen it was necessary to generate binary codes for the dataset. To achieve this, the original code of the "Attribute Discovery via Predictable Discriminative Binary Codes" paper was used. Given that the numbers of images per animal were very imbalanced in the original dataset, a decision was taken to only use 500 images per animal and to discard any animal that did not have at least 500 images.

Because of this the number of observations reduced from 30475 to 17500.

Thirdly once the dataset was processed and the binary codes were created for each of the observations, it was crucial to understand whether those "unnamed" features could be named easily and could be used to build such application.

In order to assess the suitability of those binary codes a small application was built. The application was built using the MEAN stack (MongoDB, Express, AngularJS and NodeJS). This application was simply used to get the binary codes generated and to display the images with a 1 and the images with a 0 for each of the bits.

The result was the following:

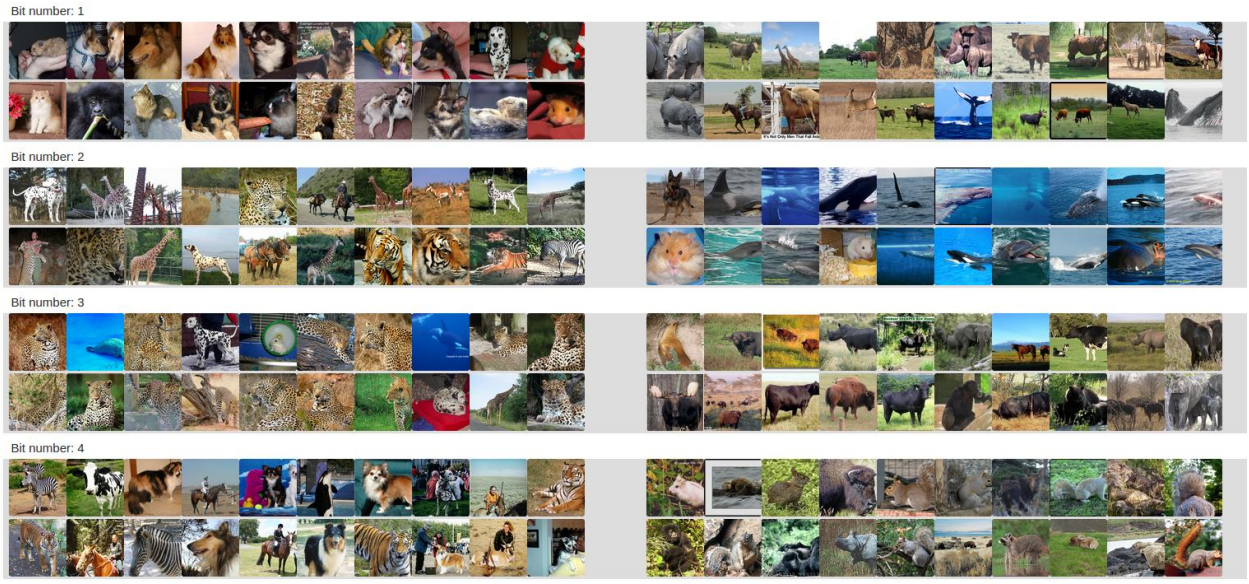


**Figure 1: Binary features of AwA (bits 1-4)**


Figure 2: Binary features of AwA (bits 5-8)

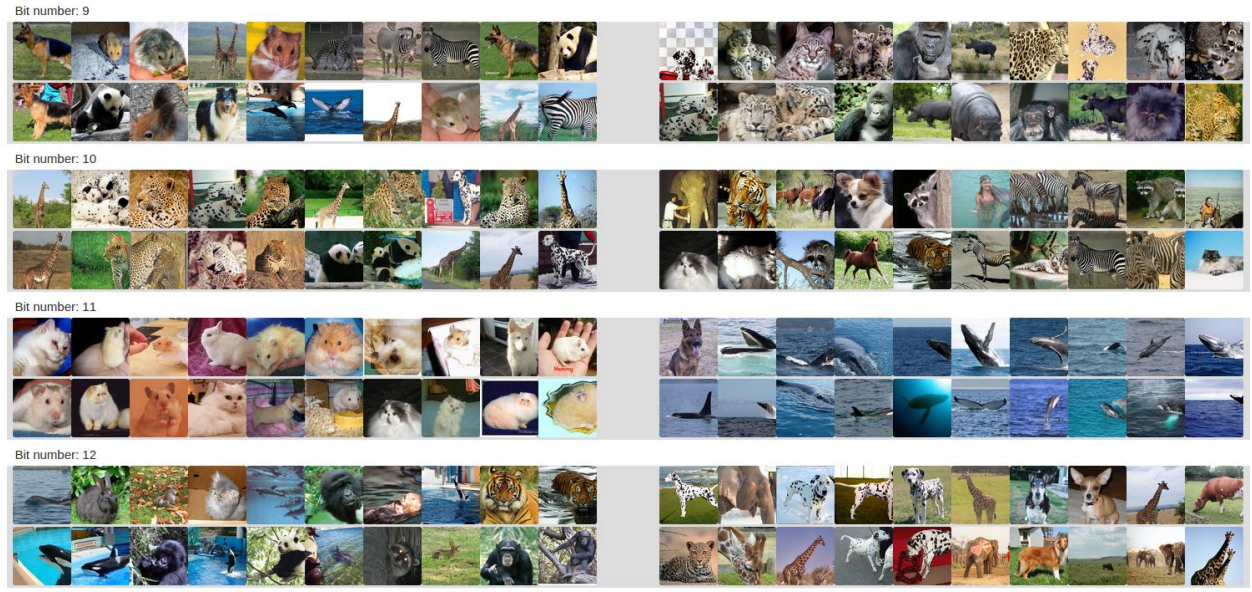


Figure 3: Binary features of AwA (bits 9-12)

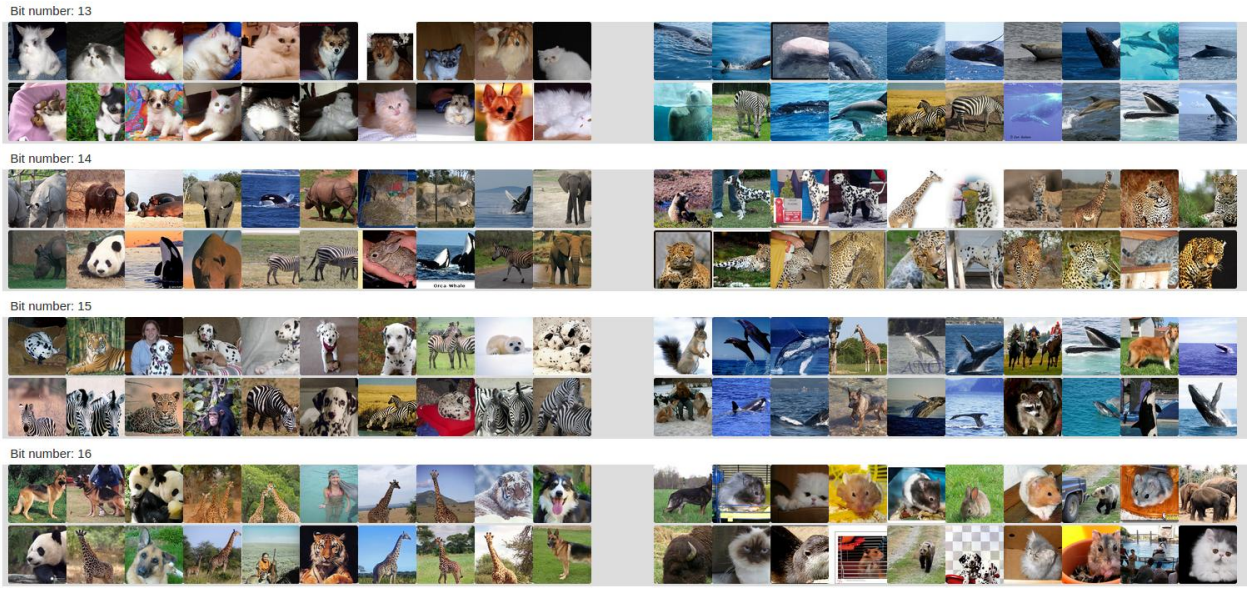


**Figure 4: Binary features of AwA (bits 13-16)**

It soon became clear that although the binary codes were indeed very discriminative and for certain bits the “unnamed” feature could be named (for example: spots or stripes), some of the bits were redundant or very close to each other.

This posed a problem given that if some of the bits were redundant or almost describing the same unnamed feature, it would be very difficult to build something that would make sense for users.

Additionally the other issue is that some of the features were less easy to name than others.

Once this realization was drawn the direction of the project steered slightly and focused specifically on the process of creating binary codes, trying to create binary codes that are not redundant but orthogonal to each other.

# 4. Generating orthogonal binary codes

In this section the method of generating binary features is briefly explained, identifying the reason some of the binary codes may be redundant and how to overcome the issue. This is followed by an explanation of the orthogonality constraint and how it was added to the initial formulation.

## A. Original method

In the original method by Rastegari et al (2012) the original problem is formulated as follows:

$$\min_{w,\xi,L,B} \frac{1}{2} \sum_{c\in\{1:C\}} \sum_{m,n\in c} d(B_m, B_n) + \gamma \sum_{s\in\{1:k\}} \|w^s\|^2$$

$$+\lambda_1 \cdot \sum_{\substack{i\in\{1:N\} \\ s\in\{1:k\}}} \xi_i^s - \frac{\lambda_2}{2} \sum_{c'\in\{1:C\}} \sum_{\substack{c''\in\{1:C\} \\ c'\neq c'', q\in c''}} d(B_p, B_q)$$

$$s.t.\ \ l_i^s(w^{s'} x_i) \geq 1 - \xi_i^s \quad \forall i \in \{1:N\}, s \in \{1:k\}$$

$$b_i^s = (1 + sign(w^{s'} x_i))/2 \quad \forall i \in \{1:N\}, s \in \{1:k\}$$

$$\xi_i^s > 0 \quad \forall i \in \{1:N\}, s \in \{1:k\}$$

"where $d$ can be any distance in the hamming space, $B_i = [b_i^1, b_i^2, \dots, b_i^k]$, $w^s$ is the weight vector corresponding to the $s^{th}$split, $\varepsilon_i^s$ is the slack variable corresponding to the $s^{th}$ split and $i^{th}$ example, $C$ is the total number of categories, $k$ is the number of splits, $N$ is the total number of examples in the train set, $l_i^s$ the training label for the $i^{th}$ example to train the $s^{th}$ split, and $b_i^s$ indicates the prediction results of $i^{th}$ example using the split $s$."(Rastegari et al, 2012)

As it can be seen from this formulation the method tries to minimize the distance between binary codes of images which are similar to each other whilst maximizing the distance between binary codes of images which are dissimilar.

Also we can see from the formulation that the only constraint on the weight vector $w$ is the regularization parameter of the margin. Additionally this constraint is imposed only on each vector $w$ for each of the $k$ splits. This means that there are constraints between the $w$ vectors of different splits.

For all we know given the original formulation two or more $w$ vectors may be representing the same visual characteristic of an image.

For this reason the next logical step was to add a constraint which would enable an easy differentiation between each of the binary code. This was achieved by adding an orthogonality constraint.

## B. Orthogonality Constraint

By definition orthogonality is a relation between two lines at right angles between each other.

This concept can also be applied to vectors, given that vectors can be thought of in the geometric form as quantities with a magnitude and a direction.

For this reason, we can describe operations between vectors in both an algebraic sense and a geometric sense.
Given this, we can express the dot product in the geometric interpretation as follows:

$$a * b = |a| * |b| * cos\theta$$

Where $\theta$ is the angle between vector $a$ and vector$b$.

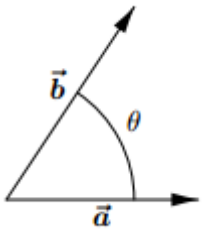


Figure 5: Geometric interpretation of 2D vectors

Looking back at the definition of orthogonality and at the definition of the dot product in the geometrical sense, we can quickly see that 2 vectors are orthogonal when their dot product is equal to 0.

This is true given that when $\theta = 90° \rightarrow cos\theta = 0 \rightarrow |a| * |b| * cos\theta = 0$

## C. Revised approach

Having looked at the definition of orthogonality the next step was to add an orthogonality constraint to the original formulation.

Firstly it was crucial to understand which of the variables would be subject to the orthogonality constraint.

As it can be seen from the original formulation:

$$b_i^s = (1 + sign(w^{s'} x_i))/2$$

Meaning that each of the binary code is a direct manipulation of the dot product between a specific weight vector $w^s$ and $x$. For this reason the orthogonality constraint should be put in place between the weight vectors $w$.

Also from the original formulation we can understand that in order to create $s$ binary codes we require $k$ number of weight vectors $w$, given that $s \in \{1:k\}$.

So given a matrix $W$ being the matrix $N * K$ where each of the columns is the $w^s$ vector, we can describe the orthogonality constraint as follows:

$$\|W^T W - I\|_{Fro}^2 = 0$$

Where $I$ is the identity matrix of the resulting $k$ by $k$ matrix from the $W^T W$ operation.

The resulting matrix of the $W^T W$ operation is a matrix $k$ by $k$ where each of the cell $i, j$ corresponds to the dot product between $w^i$ and $w^j$.

$$\begin{bmatrix} {w^1}^T w^1 & \cdots & {w^1}^T w^k \\ \vdots & \ddots & \vdots \\ {w^k}^T w^1 & \cdots & {w^k}^T w^k \end{bmatrix}$$

Essentially this operation enforces that:

$$dot(w^i, w^j) = 0$$
$$when\ i \neq j$$

This means that by adding the additional term to the original formulation, we will be constraining the optimization to ensure orthogonality between each of the weight vectors.

As such, we can define the new formulation as follows:

$$\min_{w,\varepsilon,L,B} \frac{1}{2} \sum_{c\epsilon\{1:C\}} \sum_{m,n\epsilon c} d(B_m, B_n) + \gamma \sum_{s\epsilon\{1:k\}} \|w^s\|^2 + \gamma_1 \cdot \sum_{\substack{i\epsilon\{1:N\} \\ s\epsilon\{1,k\}}} \varepsilon_i^s$$
$$- \frac{\gamma_2}{2} \sum_{\substack{c'\epsilon\{1:C\} \\ p\epsilon c'}} \sum_{\substack{c''\epsilon\{1:C\} \\ c'\neq c'', q\epsilon c''}} d(B_p, B_q) + g \cdot \|W^T W - I\|_{Fro}^2$$

$g$ Is an additional regularization parameter required for the various experiments to regularize the orthogonality constraint.

# 5. Implementation

Given that only one parameter added to the original formulation rather than implementing the entire solution from scratch the existing dbc code was modified.

In this section the details of how the code was modified are highlighted and how the SVM was implemented with the additional constraint.

## A. Original implementation

The original implementation of the optimization problem is as follows:

**Algorithm 1** Optimization

**Input:** $X = [x_1, ...x_N]$ ($x_i$ is low-level feature vector for $i^{th}$ image).
**Output:** $B$ ($B_i = [b_i^1, b_i^2, ..., b_i^k]$ is binary code for $i^{th}$ image ).

1: Initialize $B$ by: $B \leftarrow$ Projection of $X$ on first $k$ components of $PCA(X)$
2: Binarize $B$: $B \leftarrow (1 + sign(B))/2$
3: **repeat**
4: Optimizing for $B$ in $\min_B \frac{1}{2} \sum_{c\in\{1:C\}} \sum_{m,n\in c} d(B_m, B_n) - \frac{\lambda_2}{2} \sum_{c'\in\{1:C\} p\in c'} \sum_{c''\in\{1:C\} c'\neq c'', q\in c''} d(B_p, B_q)$ (see supplementary materials for details )
5: $l_i^s \leftarrow (2b_i^s - 1)\ \forall i\ i \in \{1 : N\}, \forall s\ s \in \{1 : k\}$
6: Train $k$ linear-SVMs to update $w^s\ \forall s$, $s \in \{1 : k\}$ using $L$ as training labels ($l_i^s$ : label for $i^{th}$ image when training $s^{th}$ split)
7: $b_i^s \leftarrow (1 + sign({w^s}^T x_i))/2\ \ \forall i, s\ i \in \{1 : N\},\ s \in \{1 : k\}$
8: **until** Convergence on optimization 1

As it can be seen from step 6 the original implementation trains $k$ linear-SVMs to perform the minimization on $w$ once fixed $L$ and $B$. Because the new formulation adds a constraint on $W$, modifying the code of the SVM itself was required.

The original implementation uses an off the shelf implementation of SVM called liblinear (Fan, 2008) and as the library is very comprehensive at this stage it was decided to write a custom version of SVM.

This choice was taken as it would be easier to implement the SVM algorithm specific to this optimization rather than a comprehensive library such as liblinear.

## B. SVM implementation

To put it simply SVM is just another optimization problem.

Rather than writing the optimization algorithm from scratch, a library called L-BFGS-B-C was used (Becker). The library implements a variation of the Limited-memory BFGS algorithm that uses an estimation of the inverse Hessian matrix to navigate through the search space of the optimization.

Like every other optimization problem there are 2 main elements that are required:

1. The value to optimize
2. The gradient of the objective value

In this case the value to optimize is the loss function (hinge loss is used), which is as follows:

$$L_{(W)} = \sum_{i\epsilon\{1:k\}} \left( \sum_{j\epsilon\{1:N\}} \max\left(0, 1 - y_j^i \langle w^i, x_j^i \rangle\right) + c \cdot {w^i}^T w^i \right) + g\|W^T W - I\|_{Fro}^2$$

$where$:

$$W = \begin{bmatrix} w_1^1 & \cdots & w_1^k \\ \vdots & \ddots & \vdots \\ w_M^1 & \cdots & w_M^k \end{bmatrix}; \; Y = \begin{bmatrix} y_1^1 & \cdots & y_1^k \\ \vdots & \ddots & \vdots \\ y_N^1 & \cdots & y_N^k \end{bmatrix};$$

And where $X$ consists of a 3 dimensional matrix given that there is a matrix $N$-by-$M$ for each binary code to be learnt ($k$ of them).

As it can be seen, the loss function is simply the Hinge loss followed by a regularization parameter all summed up with the orthogonality constraint described in the previous chapter.

The second element required for the optimization is the gradient of the loss function. This can be found below.

$$\nabla_{\mathrm{L_{(W)}}} = \sum_{i\epsilon\{1:k\}} \left( \sum_{j\epsilon\{1:N\}} \prod \left[ \left(1 - y_j^i \langle w^i, x_j^i \rangle\right) > 0 \right] (-y_j^i \cdot x_j^i) + 2cw^i \right) + 4g \cdot W \cdot (W^T W - I)$$

Given that the objective function and its gradient were found, a wrapper function was written to plug those functions into the optimization algorithm, together with the input data. (customSVMTrain2)

Lastly as the optimization algorithm can only work with a vector and not a matrix, the values of W were concatenated horizontally, similarly to the output of the gradient of the loss.

In order to verify that the calculations of the gradient were in fact correct, the function checkgradient was used (Rennie, 2006).

## C. Changes to original code

Once the custom SVM was implemented for this application, the original dbc code was modified.

Firstly the part of the original code that used liblinear was modified to use the custom SVM. This included more than just swapping the 1 line of code given that the inputs provided to liblinear in the original code were different from what the custom SVM requires.
To be specific, in the original code there are $k$ linear-SVMs being trained and therefore there were $k$

matrices $X$ of dimensions $N$-by-$M$ whilst the custom SVM requires the entirety of the training data, together with the associated label vectors.

Secondly the original code was modified to pass the necessary parameters to the code and to get rid of any unused variables.

# 6. Methods of evaluation

In order to understand and quantify whether adding the orthogonality constraint does in fact result in better binary codes, an effective evaluation methods was needed.

Given the nature of the problem it is difficult to estimate an accurate measure because we do not have the "correct" binary codes for each of the images. In fact, there is no "correct" binary code for each image.

Even though we do not know what the "correct" binary codes may be for those images, we have an idea of how to compare binary codes and understand which one is "better".

In order to get the understanding of what a "good" and a "bad" set of binary codes may be, we can simply interpret the original problem formulation in section 4.A.

As we can see from the original problem formulation, we are trying to minimize the distance of two binary codes when two images are similar, and to maximize the distance between two binary codes when two images are dissimilar.

Given this definition we can assume that the closer 2 binary codes are the more similar 2 images are.

Following this concept, we can also assume that by clustering on the binary codes of $N$ images, given $C$ number of classes, 2 images of the same class will be assigned to the same cluster if their binary codes are similar. We could also assume that after the clustering 2 images that belong to different classes will be in different clusters. This is the concept that was used to create a measure of effectiveness.

Essentially to get a measure of how binary codes compare the resulting binary codes are clustered. When clustering the binary codes the Hamming distance is used. This is due to the fact that we are dealing with binary data and therefore a difference of 1 bit in an 8-bit code is the same no matter the position of the bit that is different (i.e. $d_H(11111111,01111111) = d_H(11111111, 11111110)$ )

After clustering the results there are 3 main measures that are computed:

1. Adjusted Random Index (Kaijun, 2007)
2. Mutual Information (Chen, 2012)
3. Normalized Mutual Information (Chen, 2012)

Those measures are obtained by comparing the computed clustering with the original tag for each of the animals.

# 7. Experiments details

Once the original code was modified and some measures were in place to assess the performance of the algorithm, an experiment was set up to try and understand how much impact the orthogonality constraint would have.

## A. Dataset

Firstly given that the Animal with Attributes dataset was already obtained to explore the possible creation of the 20 questions game, it was decided to proceed with the same dataset.

As mentioned earlier the original dataset consists of 30475 images of 50 animals, however the number of images per animal is not balanced. For this reason 35 animals were extracted from the dataset and 500 observations per animals were picked.

This resulted in the final dataset being 17500 observations.

Secondly in the original dataset the features of each observation are in 4096 dimensions. Early on in the experiments it was clear that in order to be able to get the experiments to run in a reasonable amount of time, it was necessary to reduce the dimensionality of the data.

Some considerations were made before doing this. Given that the final aim of this experiment was to compare the performance of the algorithm with the orthogonality constraint and the algorithm without the orthogonality constraint, it was not necessary to keep as much data as possible therefore it was acceptable to reduce the dimensionality of the data.

The dimensionality of the data was reduced using PCA and it was reduced from 4096 dimensions to 100 dimensions. This was done out of necessity given that the high dimensionality and high number of observations were causing the initial experiments to take 5+ days to run.

As mentioned earlier this should not impact the results of the experiments given that the focus of this project has now become comparing the code with and without orthogonality constraint.

## B. Set of applications

During this project 2 main applications were developed.

The first application is the MATLAB application used to generate binary codes.
Essentially the MATLAB application was the main application used to gather data.

The application retrieves the Animals with Features data from file system, creates the binary codes, evaluates the results and stores the obtained codes in a MongoDB database.

The second application is simply a visualization of the binary codes itself. This application is built using NodeJS and AngularJS. The application simply serves to the client the data stored in the MongoDB database and the pictures themselves. An overview of the application architecture can be seen in the figure below.

### C. Experiments set-up

In order to understand and quantify the difference between the algorithm with the orthogonality constraint and without the orthogonality constraint the experiment was set up as follows:

1. Different sizes of binary codes were setup: 2, 4, 8, 16, 24, 32
2. Various values for the regularization parameter $c$ were assigned, from $10^{-3}$ to $10^{+3}$ (with steps of 1 for the exponent)
3. Various values for the regularization parameter $g$ were assigned ($g$ being the regularization parameter for the orthogonality constraint): 0, 1, 10
4. For each combination of the 3 parameters above, $k$-fold validation was carried out (with $k = 3$)

With this setup there should be enough data to make the experiment relevant and to give a real indication of the concrete difference between the orthogonality constraint turned on or off.

## 8. Results

After the experiment was run the data was extracted into a spreadsheet (attached).

A summary of all of the results is as follows:

| Bits | 2 | | |
|---|---|---|---|
| G | 0 | 1 | 10 |
| Avg ARI | 0.0066 ± 0.0003 | 0.0066 ± 0.0002 | 0.0063 ± 0.0002 |
| Avg NMI | 0.1773 ± 0.0029 | 0.1779 ± 0.0015 | 0.1746 ± 0.0028 |
| Avg MI | 0.3087 ± 0.0069 | 0.3101 ± 0.0035 | 0.3020 ± 0.0069 |

| Bits | 4 | | |
|---|---|---|---|
| G | 0 | 1 | 10 |
| Avg ARI | 0.0137 ± 0.0002 | 0.0127 ± 0.0014 | 0.0137 ± 0.0003 |
| Avg NMI | 0.2325 ± 0.0018 | 0.2257 ± 0.0105 | 0.2326 ± 0.0020 |
| Avg MI | 0.6107 ± 0.0064 | 0.5812 ± 0.0441 | 0.6104 ± 0.0072 |

| Bits | 8 | | |
|---|---|---|---|
| G | 0 | 1 | 10 |
| Avg ARI | 0.0140 ± 0.0012 | 0.0138 ± 0.0006 | 0.0135 ± 0.0008 |
| Avg NMI | 0.2264 ± 0.0076 | 0.2248 ± 0.0034 | 0.2226 ± 0.0046 |
| Avg MI | 0.6464 ± 0.0419 | 0.6365 ± 0.0242 | 0.6291 ± 0.0296 |

| Bits | 16 | | |
|---|---|---|---|
| G | 0 | 1 | 10 |
| Avg ARI | 0.0149 ± 0.0004 | 0.0149 ± 0.0003 | 0.0148 ± 0.0004 |
| Avg NMI | 0.2278 ± 0.0032 | 0.2274 ± 0.0024 | 0.2268 ± 0.0025 |
| Avg MI | 0.6970 ± 0.0128 | 0.6963 ± 0.0104 | 0.6935 ± 0.0110 |

| Bits | 24 | | |
|---|---|---|---|
| G | 0 | 1 | 10 |
| Avg ARI | 0.0153 ± 0.0002 | 0.0151 ± 0.0006 | 0.0153 ± 0.0003 |
| Avg NMI | 0.2295 ± 0.0018 | 0.2290 ± 0.0030 | 0.2297 ± 0.0028 |
| Avg MI | 0.7099 ± 0.0071 | 0.7061 ± 0.0158 | 0.7100 ± 0.0113 |

| Bits | | 32 | |
|---|---|---|---|
| G | 0 | 1 | 10 |
| Avg ARI | 0.0156 ± 0.0002 | 0.0155 ± 0.0002 | 0.0156 ± 0.0002 |
| Avg NMI | 0.2324 ± 0.0013 | 0.2315 ± 0.0011 | 0.2321 ± 0.0021 |
| Avg MI | 0.7234 ± 0.0053 | 0.7204 ± 0.0047 | 0.7226 ± 0.0081 |

The tables show the mean and standard deviation values of the ARI, NMI and MI for each of the G value (G being the regularization parameter of the orthogonality constraint) for each of the binary code length (2,4,8,16,24,32).

Those averages are the averages for the experiments with fixes bit length and fixed G, but with values of C going from $10^{-3}$ to $10^{+3}$.

Those results can be summarized with the following charts:

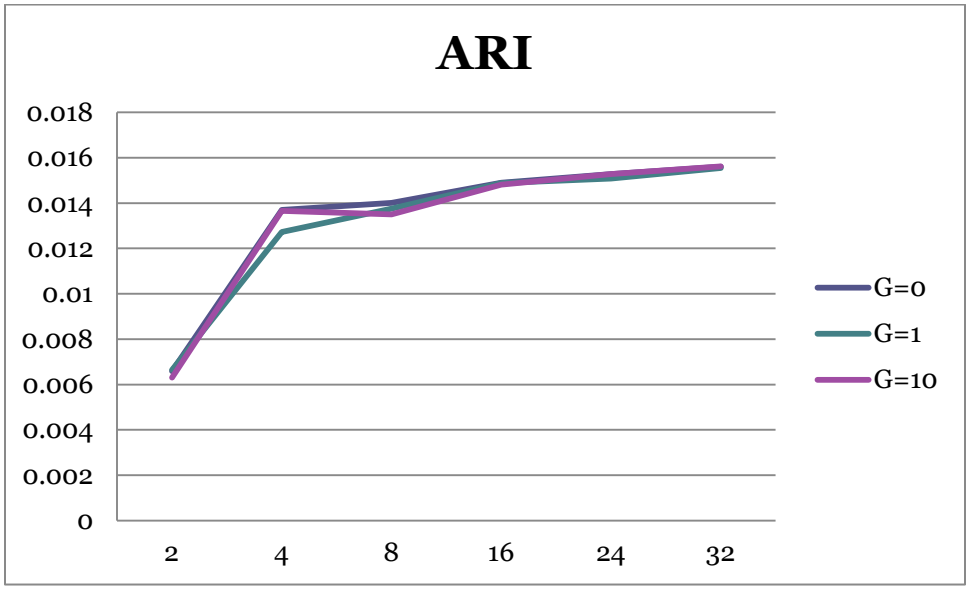


Figure 6: Results - ARI

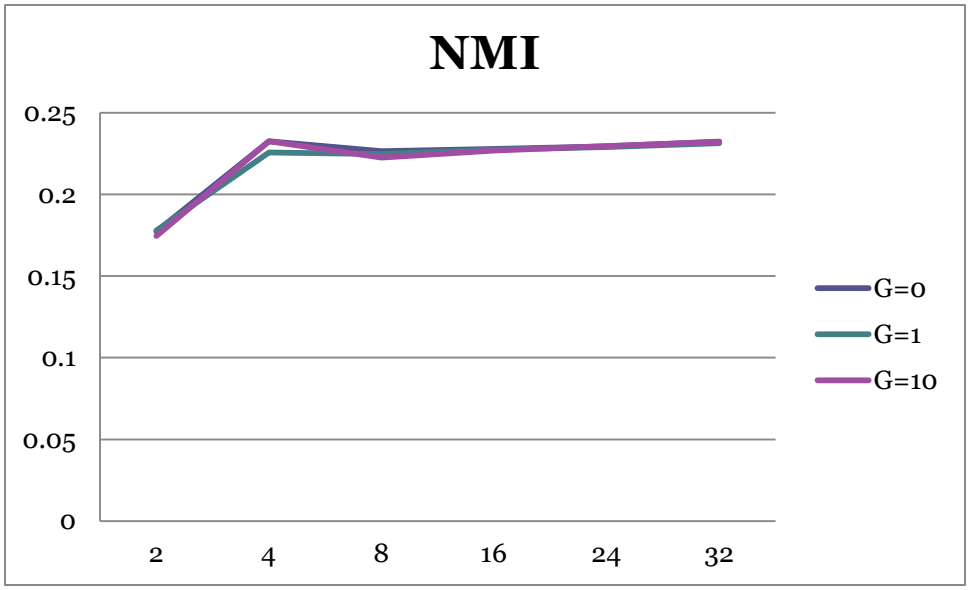


Figure 7: Results - NMI

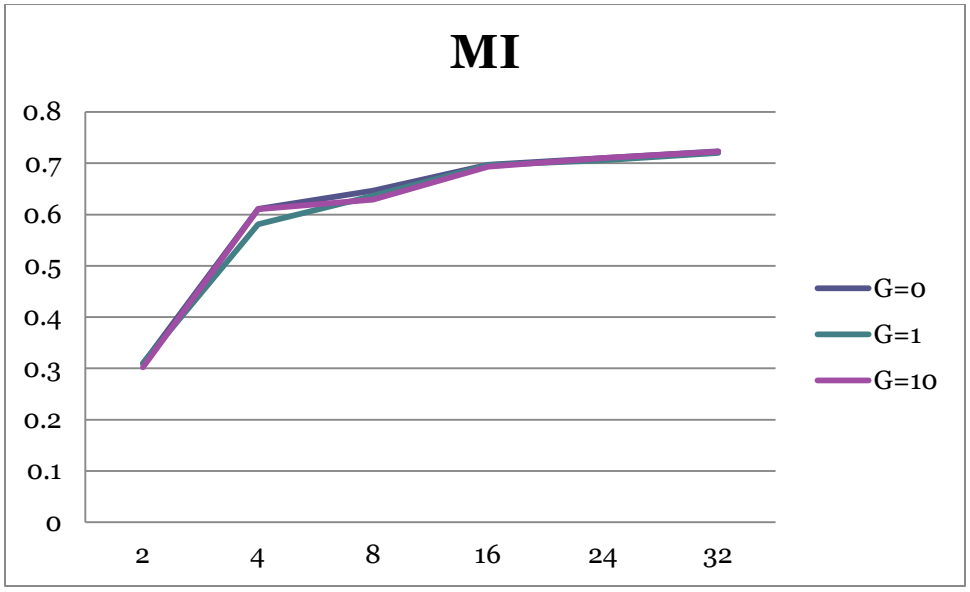


Figure 8: Results: MI

As it can quickly be gathered from the charts, adding the orthogonality constraint does not impact on the performance of the binary codes at all (according to the selected measures). This in itself is certainly an interesting result.

From this we can gather that no matter whether the orthogonality constraint is included or not, we can gain binary codes which allow us to cluster effectively.

A trend that we can easily recognize is that the accuracy goes up as the length of the binary codes goes up. This can be easily explained given that as we have more and more bits, we can recognize more and more “labels”. For example if we only have a binary code of length 2, in the best case scenario we would be able to recognize 4 different labels (animals in this instance). Similarly as the number goes up, so does the number of possible animals that we can discriminate against given the binary codes.

Once these results were gathered the next question to answer was whether the resulting binary codes are indeed more recognizable to humans and whether by adding the constraint we have in fact eliminated (or considerably reduced) the amount of redundant bits.

In order to answer this questions the binary codes were produced for the best C value with G=0 and the best C value with G=10.

Once the codes were produced on the entire dataset, the results were inserted in the database and visualized using the visualization application.

The results will be illustrated below. Please note that the visualizations should be interpreted as follows:

Each row describes a BIT, the left-hand column contains images that have the lowest values for that bit, conversely on the right-hand column contains images that have the highest values for that bit.

With orthogonality OFF (first 10 out of 32 bits):

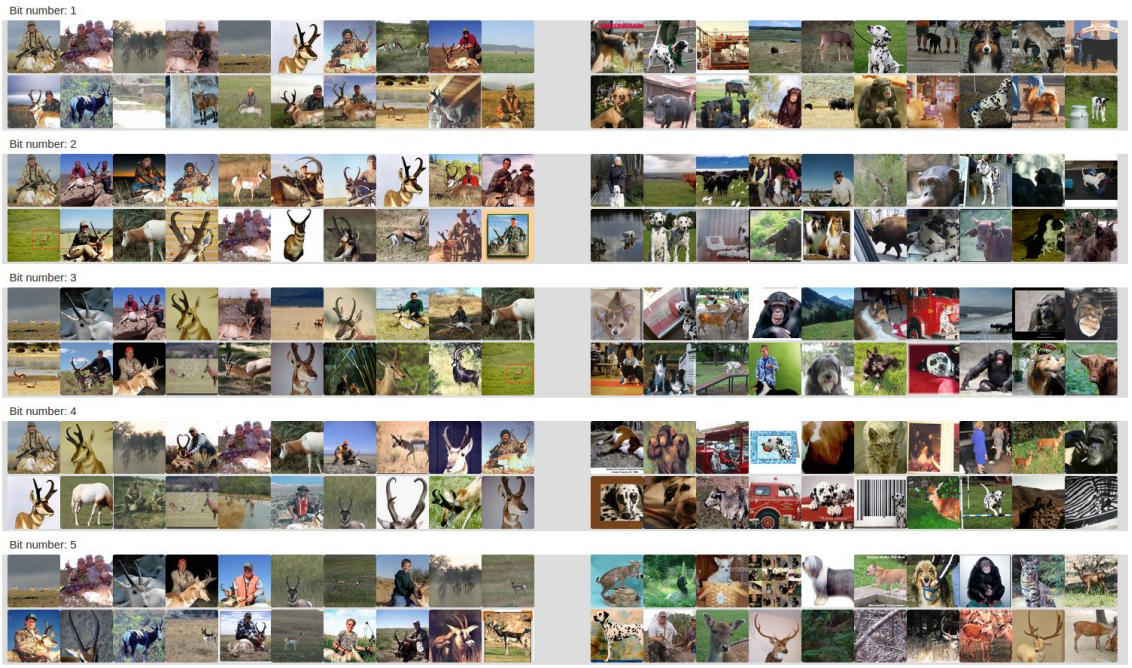


Figure 9: Orthogonality off (bits 1-5)


Figure 10: Orthogonality off (bits 6-10)

As we can see from these results, in certain occurrences we can see that multiple bits seem to encompass very similar feature about the image. In specific if we look at Bit number 6 and 7, we can see that their highest scoring images are very similar. This is exactly the type of problem that the revised formulation is trying to solve.

The following visualization displays the results with the orthogonality ON (first 10 out of 32 bits):


Figure 11: Orthogonality on (bits 1-5)

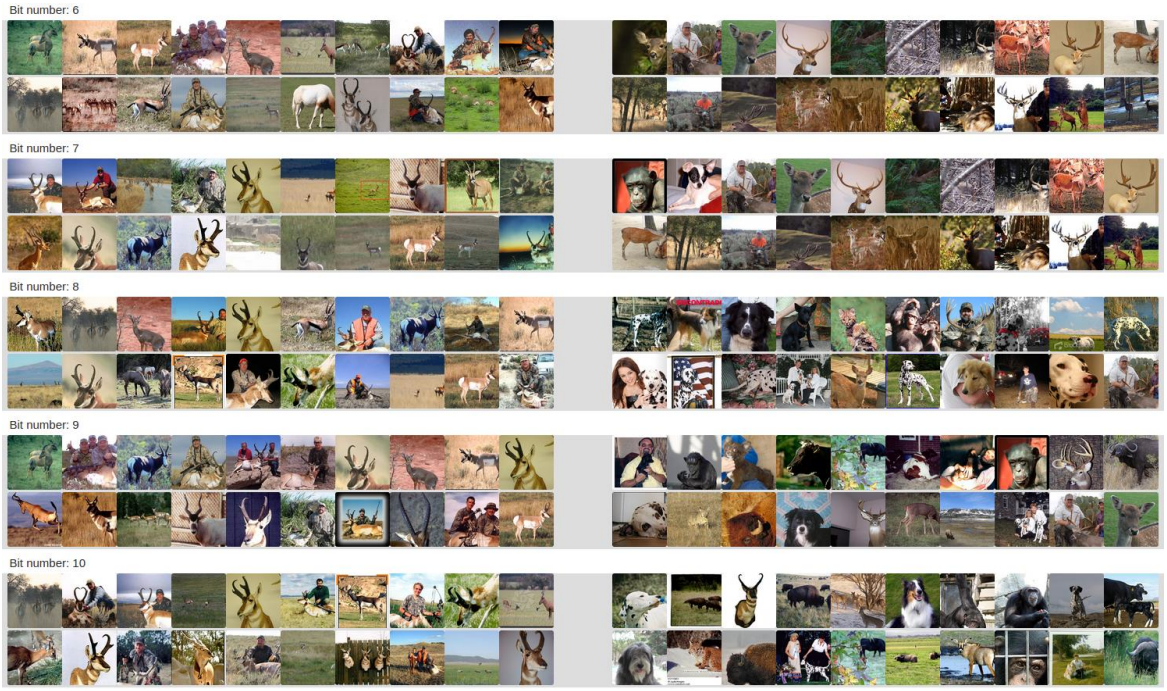


**Figure 12: Orthogonality on (bits 6-10)**

Compared to the visualization of the data without the orthogonality constraint, we can see that (in most cases) for each bit we have a very different set of images. However we can also see that in some of the bits images are repeated, for example in bits 6 and 7. This can be explained by the fact that the orthogonality constraint was indeed regularized and in order to enforce a stricter rule, the G can be set to a higher value.

The other interesting observation is that by comparing the visualizations side by side (window on the left is displaying the data obtained with orthogonality, on the right without orthogonality), we can see that some of the pictures for the low values of each bit are similar for both visualizations.

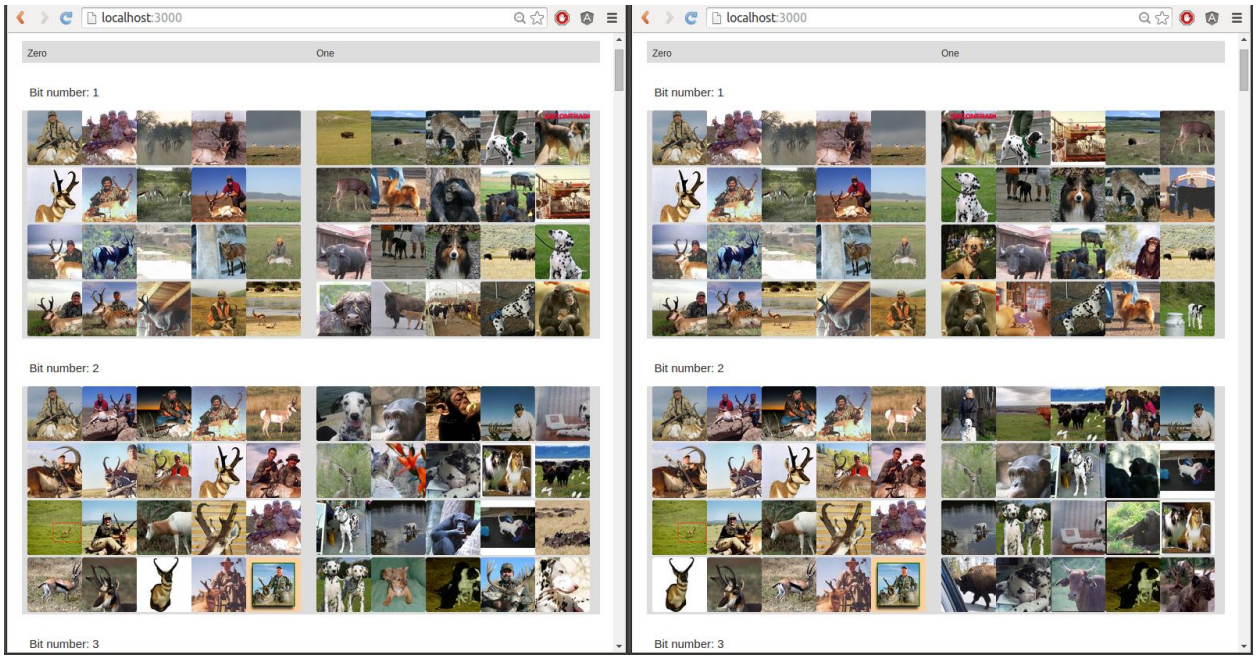


Figure 13: Results side-by-side (bits 1-2)

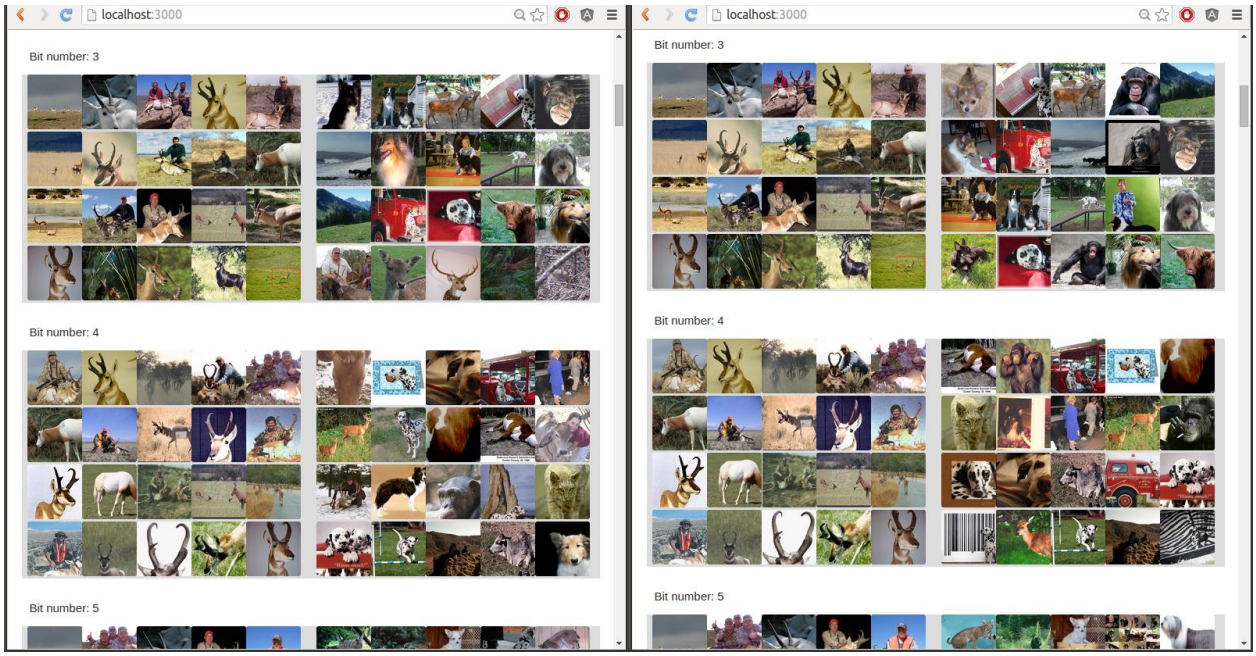


Figure 14: Results side-by-side (bits 3-4)

# 9. Discussion

Having worked on this project for a considerable amount of time, I had many chances to reflect on the main proposition of this method.

Although this method improves specifically in getting bits that encompass different features, it may be more useful in certain scenario than others. In specific, this may be very useful in tasks such as image retrievals or in the context of building a game where each feature should be fairly “unique”, however it may decrement performance in other type of applications.

One example in which this method may decrement performance is classification. The reason for this is that by forcing each of the bit to be orthogonal, we could potentially loose discriminative power for that specific bit, which may results in binary codes which are less discriminative.

# 10. Further Work

Although some experiments have been run to understand the impact of adding the orthogonality constraint, it would be interesting to carry out more experiments and possibly build an application with binary codes generated with and without the orthogonality constraint.

Given that the initial aim was to create binary features that make sense for humans, it would also be interesting to conduct experiments with groups of people that can identify and name binary codes.

Lastly there are some interesting techniques to evaluate binary codes, in this project I decided to build my own evaluation measures because of the nature of the evaluation. However it would be interesting to compare the results obtained with other evaluation methods and bigger datasets.

# 11. Bibliography

Becker, S. (n.d.). L-BFGS-B-C. Retrieved from https://github.com/stephenbeckr/L-BFGS-B-C

Chen, M. (2012, March 13). Information Theory Toolbox. Retrieved from http://uk.mathworks.com/matlabcentral/fileexchange/35625-information-theory-toolbox

Fan, R.-E. e. (2008). LIBLINEAR: A library for large linear classification. *The Journal of Machine Learning Research 9* , 1871-1874.

Gionis, A., Indyk, P., & Motwani, R. (1999). Similarity Search in High Dimensions via Hashing. *VLDB , 99*, 518-529.

Gong, Y., & Lazebnik, S. (2011). Iterative quantization: A procrustean approach to learning binary codes. *Computer Vision and Pattern Recognition (CVPR)* , 817-824.

*Instagram - Press*. (2015). Retrieved July 2015, 20, from Instagram: https://instagram.com/press/

Kaijun, W. (2007). RandIndex. Retrieved from http://uk.mathworks.com/matlabcentral/fileexchange/13916-simple-tool-for-estimating-the-number-of-clusters/content/valid_RandIndex.m

Kulis, B., & Darrell, T. (2009). Learning to hash with binary reconstructive embeddings. *Advances in neural information processing systems.* , 1042-1050.

Kulis, B., & Grauman, K. (2009). Kernelized locality-sensitive hashing for scalable image search. *IEEE 12th International Conference on Computer Vision* , 2130-2137.

Lampert, C. H., Nickisch, H., Harmeling, S., & Weidmann, J. (2008-2015). *Animals with Attributes - A dataset for Attribute Based Classification*. Retrieved 10 15, 2014, from http://attributes.kyb.tuebingen.mpg.de/

Rastegari, M., Farhadi, A., & Forsyth, D. (2012). Attribute discovery via predictable discriminative binary codes. *Computer Vision–ECCV* , 876-889.

Rennie, J. (2006, March). checkGrad2. Retrieved from http://qwone.com/~jason/matlab/checkgrad2.m

Salakhutdinov, R., & Hinton, G. (2009). Semantic hashing. *International Journal of Approximate Reasoning* , 969-978.

Shakhnarovich, G., Viola, P., & Darrell, T. (2003). Fast pose estimation with parameter-sensitive hashing. *Proceedings. Ninth IEEE International Conference on.* , 750-757.

Viola, P., & Jones, M. (2001). Rapid object detection using a boosted cascade of simple features. *Computer Vision and Pattern Recognition* , 511-518.

# 12. Appendices

## A. Matlab code

### I. Creating_label_tabel.m

```
function
label_tabel=creating_label_tabel(train_data,train_label,number_of_hypothesis)
% Initialize by PCA

[signals,PC,V] = pca2(train_data);
label_tabel=(signals(1:number_of_hypothesis,:)>=0);
```

### II. Train_hypothesis.m

```
function hypothesis = train_hypothesis(train_data,label_tabel,opt1, c, g)
%
    [m n]=size(label_tabel);
    number_of_hypothesis=m;
    low_dim=length(train_data(:,1));
    customSVM = true;
    [M N] = size(train_data);
    %custom SVM here
    posneg_train_data = zeros(N, M, number_of_hypothesis);
    posneg_train_label = zeros(N, number_of_hypothesis);
    for i=1:number_of_hypothesis
        train_label=label_tabel(i,:);
        pos_train_idx=find(train_label==1);
        neg_train_idx=find(train_label~=1);
        posneg_train_data(:,:,i)=double([train_data(:,pos_train_idx)
train_data(:,neg_train_idx)]'); % binarized verion of classemes
        posneg_train_label(:,i)=[ones(length(pos_train_idx),1); -
ones(length(neg_train_idx),1)];
    end
    hypothesis = customSVMtrain2(posneg_train_data, posneg_train_label, c,
g);
end
```

### III. Update_hypothesis.m

```
function new_hypothesis=update_hypothesis(hypothesis,data,num_of_cat,opt1, c, g)
[m n]=size(data);
%  label_tabel=(hypothesis'*[data; ones(1,n)])>0;
%  new_hypothesis=train_hypothesis(data,label_tabel);



num_examples_per_cat=n/num_of_cat;
 label_tabel=(hypothesis'*[data; ones(1,n)])>0;
 for i=1:num_of_cat
     L(:,i)=mean(label_tabel(:,(i-
```

```
1)*num_examples_per_cat+1:i*num_examples_per_cat),2)>0.5;
     label_tabel(:,(i-
1)*num_examples_per_cat+1:i*num_examples_per_cat)=repmat(L(:,i),1,num_examples_per_cat);
 end
 new_hypothesis=train_hypothesis(data,label_tabel,opt1, c, g);
```

## IV. Update_label_tabel.m

```
function
label_tabel_new=update_label_tabel(label_tabel,num_exampels_per_cat,num_examples_in_cat)
num_of_cat=length(num_examples_in_cat);
[m n]=size(label_tabel);

lambda=(num_exampels_per_cat/n);

label_tabel_new=zeros(size(label_tabel));

for i=1:num_of_cat
    offset=sum(num_examples_in_cat(1:i-1))+1;
    S0=sum((label_tabel(:,offset:offset+num_examples_in_cat(i)-1)==0),2);
    S1=sum((label_tabel(:,offset:offset+num_examples_in_cat(i)-1)==1),2);
    sum_0_cat(:,offset:offset+num_examples_in_cat(i)-
1)=repmat(S0,[1,num_examples_in_cat(i)]);
    sum_1_cat(:,offset:offset+num_examples_in_cat(i)-
1)=repmat(S1,[1,num_examples_in_cat(i)]);
end

sum_0_all=repmat(sum((label_tabel==0),2),[1,n]);
sum_1_all=repmat(sum((label_tabel==1),2),[1,n]);
gradian_0=(1-label_tabel).*(sum_1_cat-lambda*(sum_1_all-sum_1_cat));
gradian_1=(label_tabel).*(sum_0_cat-lambda*(sum_0_all-sum_0_cat));
flip_mask=(gradian_0+gradian_1)>0;
label_tabel_new=xor(flip_mask,label_tabel);
```

## V. DBC_train.m

```
function model = DBC_train(train_data,train_label,nbits, opt1, c, g)
% Learning DBC Hyperplanes
%
% Publication:
% Attribute Discovery via Predictable Discriminative Binary Codes.
%    By M. Rastegari, A. Farhadi, D. A. Forsyth.
%    In Proceeding of ECCV'2012
%
% Code is writen by Mohammad Rastegari. Report any bugs to mrastega@cs.umd.edu
% This code was modified by Michele Ricciardi as part of the MSc project
%
% Usage Syntax:
```

```
%
%  model = DBC_train(train_data, train_label, nbits, opt)
% Input:
%  train_data: MxN real value matrix : (M: number of dimensions, N: number of
samples)
%  train_label: 1xN integer value matrix: N: number of samples
%  nbits: number of DBC hyperplanes
%  opt: SVM options (default: '-B 1 -c 1 -s 1')
%
% Output:
%  model: An structure with two fields "hypothesis" and "nbits"
%
% Thanks to Jonghyun Choi for cleaning the code.

if ~exist( 'opt1' ), opt1 = []; end
number_of_hypothesis=nbits;

uni_labels=unique(train_label);
num_of_cat=length(uni_labels);

num_exampels_per_cat=sum(train_label==1);
for i=1:num_of_cat
    num_examples_in_cat(i)=length(find(uni_labels(i)==train_label));
end

[m n]=size(train_data);

label_tabel=creating_label_tabel(train_data,train_label,number_of_hypothesis);

for i=1:2
    %% Learning hypothesis(splits)
    hypothesis=train_hypothesis(train_data,label_tabel, opt1, c, g);
    if i>1
        for j=1, hypothesis=update_hypothesis(hypothesis,train_data,num_of_cat, opt1,
c, g); end
    end

    %% Producing binary features
    [m n]=size(train_data);

    binary_features_train = (hypothesis'*[train_data; ones(1,n)])>0;

    % update binary labels
    label_tabel=binary_features_train;
    for j=1:10

label_tabel=update_label_tabel(label_tabel,num_exampels_per_cat,num_examples_in_cat);
    end

end

model.hypothesis=hypothesis;
```

```
model.nbits=nbits;
```

## VI. DBC_apply.m

```
function [H, H2]=DBC_apply(data,model)
% Testing DBC Hyperplanes
%
% Original Code is writen by Mohammad Rastegari. Report any bugs to
mrastega@cs.umd.edu
% This code was modified by Michele Ricciardi as part of the MSc project

[m n]=size(data);
H2= (model.hypothesis'*[data; ones(1,n)]);
H=H2>0;
```

## VII. getBinaryFeatures.m

```
function map = getBinaryFeatures
    experimentIndex = '17';
    addpath('/home/michele/Downloads/liblinear-1.94/matlab');
    addpath('./InfoTheory');
    addpath('./L-BFGS-B-C-master/Matlab');
    disp('Reading CSV file');
    m = csvread('./data/500PA_decaf_100pca.csv');
    class = m(:, size(m,2));
    m = m(:, 1:(size(m,2)-1));
    number_of_clusters = size(unique(class),1);

    g = 1;
    values =[-3:1:3];
    k = 3;
    z = 1;
    map = [2 4 8 16 24 32];
    map = [map; zeros(size(map,2)); zeros(size(map,2)); zeros(size(map,2))];
    for mapcount=1:size(map,2)

        nbit = map(1,mapcount);
        c_matrix = [values; zeros(size(values)); zeros(size(values));
zeros(size(values))];
        for i=1:size(c_matrix,2)

            disp('Now training...');
            cval = 10^c_matrix(1,i);
            tmp = strcat('-B 1 -c ',{' '},num2str(cval,'%.6f'),' -s 3');

            %k-fold cross validation
            CVO = cvpartition(size(m,1)/number_of_clusters,'Kfold', k);
            err = zeros(CVO.NumTestSets,3);
            for j=1:CVO.NumTestSets
```

```
                trainingIdx = CVO.training(j);
                testingIdx = CVO.test(j);
                for z=2:number_of_clusters
                    trainingIdx = [trainingIdx; CVO.training(j)];
                    testingIdx = [testingIdx; CVO.test(j)];
                end
                trainingset = m(trainingIdx,:);
                trainingclasses = class(trainingIdx,:);
                testingset = m(testingIdx, :);
                testingclasses = class(testingIdx,:);
                %train
                model=DBC_train(trainingset', trainingclasses', nbit, tmp{1},
cval, g);

                %test
                [H,H2]=DBC_apply(testingset',model);
                [err(j,1) err(j,2) err(j,3)]=evaluatingFeatures(H',
testingclasses, number_of_clusters);
            end;
            disp('Average ARI, Average NMI, Average MI');
            c_matrix(2,i) = mean(err(:,1));
            c_matrix(3,i) = mean(err(:,2));
            c_matrix(4,i)   = mean(err(:,3));
            disp(c_matrix(2:4, i));
        end;
        disp(nbit);
        disp(c_matrix);
        map(2,mapcount) = max(c_matrix(2,:));
        map(3,mapcount) = max(c_matrix(3,:));
        map(4,mapcount) = max(c_matrix(4,:));
    end;
    disp(map);

    %now get binary features for best ARI, NMI and MI
%     [ bestARIvalue bestARIidx ]=max(c_matrix(2,:));
%     getFeaturesInFiles(c_matrix(1,bestARIidx), m, class, experimentIndex);
%
%     [ bestNMIvalue bestNMIidx ]=max(c_matrix(3,:));
%     if(bestNMIidx ~= bestARIidx)
%         getFeaturesInFiles(c_matrix(1,bestNMIvalue), m, class,
experimentIndex);
%     end;
%     [ bestMIvalue bestMIidx ]=max(c_matrix(4,:));
%     if((bestMIidx ~= bestARIidx) && (bestMiidx ~= bestNMIidx))
%         getFeaturesInFiles(c_matrix(1,bestMIidx), m, class,
experimentIndex);
%     end;

end
```

## VIII. getFeaturesInFiles.m

```
function getFeaturesInFiles(cvalpower, m, class, nbits, experimentIndex,g)
    cval = 10^cvalpower;
    tmp = strcat('-B 1 -c ',{' '},num2str(cval,'%.6f'),' -s 3');
    %train
    model=DBC_train(m', class', nbits, tmp{1},cval,g);

    %test
    [H,H2]=DBC_apply(m',model);
    disp(strcat('Saving to CSV file ../results/',experimentIndex,'_',date,'-
learnedLabels_cp',num2str(cvalpower),'.csv'));
    binary_filename = strcat('../results/',experimentIndex,'_',date,'-
learnedLabels_binary_cp',num2str(cvalpower),'.csv');
    csvwrite(binary_filename, H');

csvwrite(strcat('/home/michele/Uni_Masters/project/results/',experimentIndex,'_',date,'-
learnedLabels_cp',num2str(cvalpower),'.csv'), H2');
    %TODO- write to mongodb
end
```

## IX. evaluatingFeatures.m

```
function [ARI NMI MI] = evaluatingFeatures(results, true_labels,
number_of_clusters)
    %results = csvread(filename);
    %true_labels = csvread('../results/truelabels.csv');
    [idx, c] = kmeans(results,
number_of_clusters,'dist','Hamming','start','sample',
'emptyaction','singleton');
    disp(min(idx));
    disp(max(idx));
    ARI = RandIndex(true_labels,idx);
    NMI = nmi(true_labels, idx);
    MI = mutualInformation(true_labels, idx);
    disp(ARI);
    disp(NMI);
    disp(MI);
end
```

## X. copytodb.sh

```
#! /bin/bash

csvtool paste 500PA_mappingfile_all.csv $1 >temp.csv
mongoimport -d $2 -c animals --type csv --file temp.csv --fieldFile $3
```

## XI. customSVMTrain2.m

```
function [ finalW ] = customSVMtrain2(x, trueY, c, g)
```

```
    % x is a 3 dimensional array where the last dimension indicates which
    % of the set is being targeted

    % y is 2 dimension, each row at index N is an array of labels
    % associated with the respective x matrix whose 3rd dimension is N
    if(~exist('g'))
        g=1;
    end;

    [N M D] = size(x); % M= numb of features, N=numb of datapoints, D=numb of
dimensions
    temp = zeros(N, M+1, D);
    % Add bias
    for d=1:D
        temp(:,:,d)= [x(:,:,d) ones(N,1)];
    end
    clearvars x;
    x = temp;
    clearvars temp;
    M = M+1;
    l = -inf(M*D,1);
    u = inf(M*D,1);
    I = eye(D,D);
    wcwc = zeros(D, D);
    f = @error; %returns error
    grad = @gradient; %returns gradient of error

    margin = 1;
    tic();
    %opts = struct('factr', 1e12);
    [finalW,finalF,info] = lbfgsb( {f,grad} , l, u); %Perform minimization
    toc();
    finalW = vec2mat(finalW,D);

    %loss function = sum(sum(max(0, margin-y<w,x>))+c*wTw)+ ||WcTWc-I||Fro^2
    function value = error(wc)
        value = 0;
        w = vec2mat(wc,M)';
        for i=1:D
            dotproduct = x(:,:,i)*w(:,i);
            misclas = trueY(:,i).*dotproduct;
            tmp = sum(max(0, margin-misclas))+c*w(:,i)'*w(:,i);
            value = value + tmp;
        end
        wcwc = w'*w-I;
        value = value +g*((norm(wcwc,'fro'))^2); %add Identity matrix and
extra parameter for regularization
    end

    %gradient of loss = sum(sum( (1-y<w,x>)>0 (-y*x):0)+2*c*w)+4*(WcTWc-I)*Wc
    function dobj = gradient(wc)
        dHinge = [];
```

```
        w = vec2mat(wc,M)';
        for i=1:D
            tmp=0;
            for j=1:N
                dotproduct = x(j,:,i)*w(:,i);
                if(margin-(trueY(j,i)*dotproduct) >0)
                    tmp= tmp+(-trueY(j,i)*x(j,:,i));
                end
            end
            dHinge = [dHinge; (tmp' + 2*c*w(:,i))];
        end
        wcwc = w'*w - I;
        dobj = dHinge +reshape((g*4*w*wcwc),1,[])';
    end
end
```

## B. Visualisation application

### I. server.js (backend)

```
//
// # SimpleServer
//
//
var http = require('http');
var path = require('path');

var express = require('express');
// Retrieve
var Db = require('mongodb').Db,
MongoClient = require('mongodb').MongoClient,
Server = require('mongodb').Server,
ReplSetServers = require('mongodb').ReplSetServers,
ObjectID = require('mongodb').ObjectID,
Binary = require('mongodb').Binary,
GridStore = require('mongodb').GridStore,
Grid = require('mongodb').Grid,
Code = require('mongodb').Code,
BSON = require('mongodb').pure().BSON,
assert = require('assert');
var actualDb = null;
var dbname = '17_withoutorth';

// Connect to the db
var connectToDb = function(){
  MongoClient.connect("mongodb://localhost:27017/"+dbname, function(err, db)
{
    if(!err){
      console.log("connected!");
      actualDb = db;
```

```
    }else console.log(err);
  });
};

connectToDb();
var app = express();
var server = http.createServer(app);

app.get('/data/:id/:numPic/:desc', function(req, res) {
        //Get stuff
        console.log('ID: %d\nValues: %d\nDesc:
%d',req.param('id'),req.param('numPic'),req.param('desc'));
  var options = {};
  options.limit = req.param('numPic');
  var desc = req.param('desc');
  var actualField = 'field'+req.param('id');
  if(desc == 0){
    options.sort = [[actualField,'asc']];
  }else{
    options.sort = [[actualField,'desc']];
  }
  actualDb.collection('animals').find({}, options).toArray(function(err,
obj){
    res.send(200, {'data': obj, 'index': req.param('id')});
  });
});

app.get('/changedb/:dbname', function(req,res){
  console.log('Changing to DB: ', req.param('dbname'));
  dbname = req.param('dbname');
  connectToDb();
  res.send(200);
})

app.use(express.static(path.resolve(__dirname, 'app')));
server.listen(3000, "127.0.0.1", function(){
  console.log('Listening on port %d', server.address().port);
});
```

## II. controllers.js (front-end)

```
'use strict';

/* Controllers */

var datavisapp = angular.module('dataVizApp', []);

datavisapp.controller('AwAFeaturesCtrl', function($scope, $http) {
  $scope.binaryFeatures = 32;
  $scope.picPerFeature = 20;
```

```
  $scope.features = [];
  $scope.addToZeros = function(data, index){
    for(var i in data){
      $scope.features[index-1].zero.push(data[i]);
    }
  };
  $scope.addToOnes = function(data, index){
    for(var i in data){
      $scope.features[index-1].one.push(data[i]);
    }
  };

  $scope.getAllTheData = function(){
    $scope.features = [];
    //For each feature
    for(var index=1; index<=$scope.binaryFeatures; index++){

      var finalObjectForFeature = {id: index, zero: [], one: []};
      $scope.features.push(finalObjectForFeature);

      //grab the 0s
      console.log('data/'+index+'/'+$scope.picPerFeature+'/0');

$http.get('data/'+index+'/'+$scope.picPerFeature+'/0').success(function(data)
{
        $scope.addToZeros(data.data, data.index);
      });

      //grab the 1s

$http.get('data/'+index+'/'+$scope.picPerFeature+'/1').success(function(data)
{
        $scope.addToOnes(data.data, data.index);
      });
      //compile them together
      console.log("some Data: "+index);
    }
  };
  $scope.getAllTheData();
});
```

### III. index.html (front-end)

```
<!doctype html>
<html lang="en" ng-app="dataVizApp">
<head>
  <meta charset="utf-8">
  <title>Visualising binary features</title>
  <link rel="stylesheet" href="components/bootstrap/dist/css/bootstrap.css">
  <link rel="stylesheet" href="css/app.css">
```

```
  <script src="components/angular/angular.js"></script>
  <script src="js/controllers.js"></script>
</head>
<body ng-controller="AwAFeaturesCtrl">
  <div class="container">
    <h1>Visualising binary features</h1>
    <br>
    <h2>Select the number of pictures per binary feature
      <select ng-model="picPerFeature" ng-change="getAllTheData()">
        <option default>5</option>
        <option>10</option>
        <option>20</option>
        <option>30</option>
        <option>40</option>
        <option>50</option>
        <option>75</option>
        <option>100</option>
        <option>150</option>
      </select>
      <input type="checkbox" ng-model="showfeaturesNo" value="true">
      <div ng-show="showfeaturesNo">
        Change number of features showing:
        <select ng-model="binaryFeatures" ng-change="getAllTheData()">
          <option>2</option>
          <option>4</option>
          <option default>8</option>
          <option>16</option>
          <option>24</option>
          <option>32</option>
        </select>
      </div>
      <!-- for each feature -->
      <div class="row">
        <div class="col-md-6">
          <h3>Zero</h3></div>
          <div class="col-md-6">
            <h3>One</h3></div>
          </div>
            <div class="row" ng-repeat="feature in features">
              <div class="row" ng-if="showfeaturesNo">
                <h2>Bit number: {{feature.id}}</h2>
              </div>
              <!-- show the 0s -->
              <div class="col-md-6">
                <div ng-repeat="image in feature.zero" ng-
style="{'background-image': 'url(img/' + image.pictureTag + '.jpg)'}"
class="img-rounded">
                </div>
              </div>

              <!-- show the 1s -->
              <div class="col-md-6">
```

```
                <div ng-repeat="image in feature.one" ng-style="{'background-
image': 'url(img/' + image.pictureTag + '.jpg)'}"  class="img-rounded">
                </div>
              </div>

            </div>

        </div>
      </div>
    </body>
    </html>
```

## IV. app.css

```
/* app css stylesheet */

body {
  padding-top: 20px;
}

.container{
        width: 100%;

}

.img-rounded{
        max-height: 140px;
        height: 140px;
        width: 140px;
        display: inline-block;
        background-size: cover;
    background-repeat: no-repeat;
}

.row{
        margin: 20px;
}

.col-md-6{
        background: #ddd;
}
```