Performance of Machine Learning Algorithms When Classifying Children's Clothes According to Gender

Tony Rönnqvist and Simon Westberg

***Abstract*—In this paper, we investigate a machine's ability to classify children's clothes according to gender. This was done by implementing three different machine learning algorithms; kernel ridge regression, regularized extreme learning machine, and support vector machine. A Gaussian radial basis function kernel was used for both ridge regression and support vector machine, and for extreme learning machine the softplus function was used as activation. The algorithms were trained and tested on a data set consisting of one thousand images gathered from the Swedish clothing-retail company H&M. The clothes were categorized as being for children from the ages of eighteen months to ten years.**

We found that support vector machine had the best performance on the data set and achieved a classification accuracy of 76.9%. However, the other two methods obtained similar accuracies; 76.6% for kernel ridge regression and 76.7% for regularized extreme learning machine.

***Index Terms*—children's clothes, extreme learning machine, gender classification, kernel method, machine learning, ridge regression, support vector machine**

I. Introduction

The use of machine learning algorithms for solving pattern recognition problems has greatly increased in recent years. This is partly a consequence of the increased need to properly process and analyze the large amount of data that is generated daily in our society. One of the most common problems within the area of pattern recognition is that of classifying different kinds of data. One such classification problem is to predict the gender of a person based on e.g. facial traits and clothing. Some studies have suggested that most children older than two years can correctly identify the gender of other children and correctly label objects that are typically associated as being either female or male [1]. Therefore, gender classification can be considered easy from a human perspective. It would thus be interesting to examine if a machine can develop the same abilities as a child and learn to classify children's clothes by gender. Furthermore, the topic of gender stereotypes is a current discussion in today's society, and to what extent a machine inherits these traits might also be of interest.

The aim of this project is to determine if machine learning algorithms are able to classify children's clothes by gender, and if so, to what accuracy. To this end, we implement three common algorithms; kernel ridge regression [2] [3], regularized extreme learning machine [4] [5], and support vector machine [6]. The algorithms are trained and tested on a data set consisting of 1000 images of children's clothes gathered from the Swedish clothing-retail company H&M. Some prior research has been done on gender classification in general. In the paper [7] by Cai *et al.*, they study clothing based gender recognition. Other studies include [8] and [9] where the gender classification is based on clothing as well as other attributes, such as facial expressions and hairstyles.

The paper is structured as follows. In Section II the notation used in the paper is described. Section III covers the necessary theory behind the classification problem and the different algorithms, whereas Section IV explains the project's methodology. Finally, in Sections V-VII the project's results are presented, discussed, and summarized.

II. Notation

Throughout the paper the following notation is used. Vectors are written with bold, lowercase characters, e.g. $\mathbf{x}$, whereas matrices are written with bold, uppercase characters, e.g. $\mathbf{A}$. The transpose of a vector or matrix is written as $\mathbf{A}^T$, and the Euclidean norm of a vector $\mathbf{x} = (x_1, ..., x_n)^T$ as $\|\mathbf{x}\|_2 = (x_1^2 + ... + x_n^2)^{1/2}$. For a matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$, the entry in the i-th row and the j-th column is denoted by $(\mathbf{A})_{i,j}$ and the Frobenius norm is defined by

$$\|\mathbf{A}\|_F = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n} |(\mathbf{A})_{i,j}|^2}, \tag{1}$$

where $|\cdot|$ denotes the absolute value. Finally, the trace of an $n \times n$ matrix $\mathbf{A}$ is defined as $\mathrm{tr}(\mathbf{A}) = \sum_{i=1}^{n} (\mathbf{A})_{i,i}$.

III. Theory

A. Classification

The aim of the classification problem is to assign an input vector $\mathbf{x} \in \mathbb{R}^D$ to one of k classes, $\mathcal{C}_1, ..., \mathcal{C}_k$. The class $\mathcal{C}_i$ can be represented by a target vector $\mathbf{t} \in \mathbb{R}^k$, whose entries are all zero, except for a single one in the i-th position, i.e.

$$\mathbf{t} = (0, 0, ..., \underset{i}{1}, ..., 0, 0)^T. \tag{2}$$

An input vector $\mathbf{x}$ can then be assigned to one of the classes by finding a function $\mathbf{f} : \mathbb{R}^D \longrightarrow \mathbb{R}^k$ that maps $\mathbf{x}$ to one of the target vectors $\mathbf{t}$. $\mathbf{f}$ can be modeled using a set $D_{tr} = \{\mathbf{x}_1, ..., \mathbf{x}_N\}$ of N training samples that are labeled by class, together with their corresponding target vectors, $\mathbf{t}_1, ..., \mathbf{t}_N$, by using different machine learning techniques. A new input $\mathbf{x}$ is then predicted to belong to one of the classes by calculating $\mathbf{f}(\mathbf{x})$.

B. Least squares

In a least squares model, a linear relationship is assumed between each entry in the target $\mathbf{t}$ and the input $\mathbf{x}$,

$$t_i = w_{0,i} + w_{1,i}x_1 + w_{2,i}x_2 + ... + w_{D,i}x_D, \tag{3}$$

with weights $w_{0,i}, ..., w_{D,i}$, where $i = 1, ..., k$ [2]. If we add a single one at the start of $\mathbf{x}$, so that $\mathbf{x} = (1, x_1, ..., x_D)^T$, (3) can be written as

$$\mathbf{t} = \mathbf{W}^T\mathbf{x}, \tag{4}$$

where

$$\mathbf{W} = \begin{bmatrix} w_{0,1} & \dots & w_{0,k} \\ \vdots & \ddots & \vdots \\ w_{D,1} & \dots & w_{D,k} \end{bmatrix} \in \mathbb{R}^{(D+1)\times k}.$$

The N training samples and target vectors can be collected in two matrices

$$\mathbf{X} = \begin{bmatrix} 1 & x_{1,1} & \dots & x_{1,D} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{N,1} & \dots & x_{N,D} \end{bmatrix} \in \mathbb{R}^{N\times(D+1)}$$

and

$$\mathbf{T} = \begin{bmatrix} t_{1,1} & \dots & t_{1,k} \\ \vdots & \ddots & \vdots \\ t_{N,1} & \dots & t_{N,k} \end{bmatrix} \in \mathbb{R}^{N\times k},$$

to produce a single equation for all training samples,

$$\mathbf{T} = \mathbf{X}\mathbf{W}. \tag{5}$$

Since $\mathbf{X}$ is usually not invertible, one seeks to determine $\mathbf{W}$ so that the sum of squared errors,

$$E(\mathbf{W}) := \sum_{i=1}^{N} \|\mathbf{t}_i - \mathbf{W}^T\mathbf{x}_i\|_2^2 = \sum_{i=1}^{N}\sum_{j=1}^{k} |(\mathbf{T} - \mathbf{X}\mathbf{W})_{i,j}|^2 = \\ = \|\mathbf{T} - \mathbf{X}\mathbf{W}\|_F^2, \tag{6}$$

for each data point is minimal. The least squares problem can thus be stated as a minimization problem

$$\arg\min_{\mathbf{W}} \|\mathbf{T} - \mathbf{X}\mathbf{W}\|_F^2. \tag{7}$$

The error function (6) can be expanded using the trace of $\mathbf{T} - \mathbf{X}\mathbf{W}$ [10],

$$E(\mathbf{W}) = \mathrm{tr}((\mathbf{T} - \mathbf{X}\mathbf{W})(\mathbf{T} - \mathbf{X}\mathbf{W})^T) = \\ = \mathrm{tr}(\mathbf{T}\mathbf{T}^T - \mathbf{T}\mathbf{W}^T\mathbf{X}^T - \mathbf{X}\mathbf{W}\mathbf{T}^T + \mathbf{X}\mathbf{W}\mathbf{W}^T\mathbf{X}^T). \tag{8}$$

Differentiating (8) with respect to $\mathbf{W}$, using the identities from [10], and setting the result equal to zero, we get

$$\frac{\partial E(\mathbf{W})}{\partial \mathbf{W}} = 2(\mathbf{X}^T\mathbf{X}\mathbf{W} - \mathbf{X}^T\mathbf{T}) = \mathbf{0} \implies \\ \hat{\mathbf{W}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{T} = \mathbf{X}^\dagger\mathbf{T}, \tag{9}$$

where $\mathbf{X}^\dagger$ is the Moore-Penrose inverse [10] of $\mathbf{X}$ and $\hat{\mathbf{W}}$ is the solution to (7). A prediction for a new input $\mathbf{x}$ can then be calculated using

$$\mathbf{y}(\mathbf{x}) = \hat{\mathbf{W}}^T\mathbf{x} = (y_1, y_2, ..., y_k)^T, \tag{10}$$

and using the decision function

$$\mathbf{f}(\mathbf{y}(\mathbf{x})) = (0, 0, ..., \underset{m}{1}, ..., 0, 0)^T, \tag{11}$$

such that y_m is the maximal element of $\mathbf{y}(\mathbf{x})$. The input $\mathbf{x}$ is then predicted to belong to class $\mathcal{C}_m$.

The minimal solution (9) assumes that $\mathbf{X}^T\mathbf{X}$ is invertible, i.e. that $\mathbf{X}$ has linearly independent columns. If $D > N$ this is not the case and one has to either reduce the input dimension or add a regularization term [2], as explained in Section III-C.

The least squares solution (9) can also be derived as the maximum likelihood solution under an assumed Gaussian noise model. See [3] for details.

C. Regularization

An overfitted model is a model that is too closely tuned to the training data and thus may make poor predictions on new data. To prevent overfitting, one can add a so called *regularization term* to the error function (6) that penalizes the size of the model parameters $\mathbf{W}$, as described in [2] and [3]. A common choice of regularizer is $\|\mathbf{W}\|_F^2$. The error function then becomes

$$E(\mathbf{W}) = \|\mathbf{T} - \mathbf{X}\mathbf{W}\|_F^2 + \lambda\|\mathbf{W}\|_F^2, \tag{12}$$

where $\lambda \geq 0$ is a *hyperparameter*, i.e. a model parameter that is not determined in the training phase but is instead set prior to training. The size of λ governs the importance of the regularization term.

Differentiating (12) with respect to $\mathbf{W}$, we obtain

$$\frac{\partial E(\mathbf{W})}{\partial \mathbf{W}} = 2(\mathbf{X}^T\mathbf{X}\mathbf{W} - \mathbf{X}^T\mathbf{T}) + 2\lambda\mathbf{W}. \tag{13}$$

Setting the result equal to zero gives a new expression for the least squares solution (9),

$$2(\mathbf{X}^T\mathbf{X}\hat{\mathbf{W}} - \mathbf{X}^T\mathbf{T}) + 2\lambda\hat{\mathbf{W}} = \mathbf{0} \implies \tag{14}$$

$$\hat{\mathbf{W}} = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{T}, \tag{15}$$

where $\mathbf{I}$ is the identity matrix. The addition of the regularization term to the error function thus results in adding a positive constant λ to the diagonal of $\mathbf{X}^T\mathbf{X}$ in (9). This addition leads to a modified matrix $(\mathbf{X}^T\mathbf{X}+\lambda\mathbf{I})$ in (15) which is invertible, even if $\mathbf{X}^T\mathbf{X}$ has linearly dependent columns [2]. The regularized least squares model is known as *ridge regression*.

D. Kernel trick

Using equation (14) we can reformulate the least squares solution to the so called *dual formulation* [3]. Equation (14) implies that

$$\hat{\mathbf{W}} = \frac{1}{\lambda}\mathbf{X}^T(\mathbf{T} - \mathbf{X}\hat{\mathbf{W}}). \tag{16}$$

We now define a new matrix $\mathbf{A} = \frac{1}{\lambda}(\mathbf{T} - \mathbf{X}\hat{\mathbf{W}})$, so that

$$\hat{\mathbf{W}} = \mathbf{X}^T\mathbf{A}. \tag{17}$$

The error function (12) can now be expressed in terms of $\mathbf{A}$ as

$$E(\mathbf{A}) = \|\mathbf{T} - \mathbf{X}\mathbf{X}^T\mathbf{A}\|_F^2 + \lambda\|\mathbf{X}^T\mathbf{A}\|_F^2. \tag{18}$$

Differentiating (18) with respect to $\mathbf{A}$, using the identities from [10], and setting the result equal to zero, we obtain

$$\frac{\partial E(\mathbf{A})}{\partial \mathbf{A}} = 2(\mathbf{X}\mathbf{X}^T\mathbf{X}\mathbf{X}^T\mathbf{A} - \mathbf{X}\mathbf{X}^T\mathbf{T} + \lambda\mathbf{X}\mathbf{X}^T\mathbf{A}) = \mathbf{0} \implies$$
$$\hat{\mathbf{A}} = (\mathbf{X}\mathbf{X}^T + \lambda\mathbf{I})^{-1}\mathbf{T}. \tag{19}$$

As we can see, in the dual formulation (19) an $N \times N$ matrix has to be inverted, as opposed to (15) where a $(D+1)\times(D+1)$ matrix is inverted. When the dimension D of the input vectors $\mathbf{x}$ is much larger than the number of training samples N, the dual formulation is thus less computationally heavy.

Defining a *kernel function* $k(\mathbf{x}, \mathbf{x}') = \mathbf{x}^T\mathbf{x}'$ between two input vectors $\mathbf{x}$ and $\mathbf{x}'$, as well as a kernel matrix $\mathbf{K} = \mathbf{X}\mathbf{X}^T$, so that

$$\mathbf{K} = \begin{bmatrix} k(\mathbf{x}_1, \mathbf{x}_1) & \dots & k(\mathbf{x}_1, \mathbf{x}_N) \\ \vdots & \ddots & \vdots \\ k(\mathbf{x}_N, \mathbf{x}_1) & \dots & k(\mathbf{x}_N, \mathbf{x}_N) \end{bmatrix},$$

equation (19) can be written as

$$\hat{\mathbf{A}} = (\mathbf{K} + \lambda\mathbf{I})^{-1}\mathbf{T}. \tag{20}$$

A new expression for (10) can then be made using (17) and (20),

$$\mathbf{y}(\mathbf{x}) = \mathbf{T}^T(\mathbf{K} + \lambda\mathbf{I})^{-1}\mathbf{k}(\mathbf{x}), \tag{21}$$

where $\mathbf{k}(\mathbf{x}) = (k(\mathbf{x}_1, \mathbf{x}), ..., k(\mathbf{x}_N, \mathbf{x}))^T$. A prediction is then made for a new input $\mathbf{x}$ with (11).

Another advantage of the dual formulation comes from the fact that any kernel function $k(\mathbf{x}, \mathbf{x}')$ that defines an inner product in some vector space can be used in the kernel matrix $\mathbf{K}$ and kernel vector $\mathbf{k}(\mathbf{x})$ [3]. Using ridge regression together with a kernel is known as *kernel ridge regression* (KRR).

Some common kernels include the polynomial kernel of degree d,

$$k(\mathbf{x}, \mathbf{x}') = (\gamma\mathbf{x}^T\mathbf{x}' + c)^d, \tag{22}$$

and the Gaussian radial basis function (GRBF) kernel

$$k(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|_2^2}{s}\right), \tag{23}$$

where $\gamma > 0, c > 0, d \in \mathbb{N}$, and $s > 0$ are hyperparameters.

E. Extreme learning machine

In this section, we will briefly explain the algorithm known as extreme learning machine (ELM), as proposed by Huang *et al.* [4], based on a single-hidden layer feedforward neural network (SLFN). An SLFN consists of one hidden layer with $\tilde{N}$ nodes, together with an input and output layer with D and k nodes, respectively, where D is the dimension of the input $\mathbf{x}$ and k is the dimension of the target. For each $i = 1, ..., \tilde{N}$, there is an associated weight vector $\mathbf{w}_i \in \mathbb{R}^D$ and *bias* b_i that connects the input to the i-th hidden node, together with a weight vector $\boldsymbol{\beta}_i \in \mathbb{R}^k$ that connects the same node to the output. A visual representation of such a network is shown in Fig. 1.

The SLFN is then modelled as

$$\mathbf{y}(\mathbf{x}) = \sum_{i=1}^{\tilde{N}} \boldsymbol{\beta}_i g(\mathbf{w}_i^T\mathbf{x} + b_i), \tag{24}$$

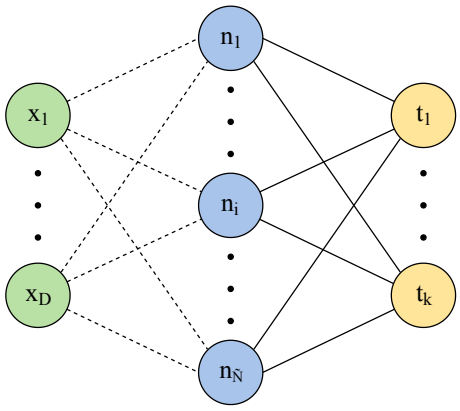

Fig. 1. Schematic overview of an SLFN with $\tilde{N}$ hidden neurons $n_1, ..., n_{\tilde{N}}$. The dashed connections represents the input weights $\mathbf{w}_1, ..., \mathbf{w}_{\tilde{N}}$ and bias' $b_1, ..., b_{\tilde{N}}$, whereas the black connections represents the output weights $\boldsymbol{\beta}_1, ..., \boldsymbol{\beta}_{\tilde{N}}$.

where $g(\cdot)$ is the so called *activation function*.

Introducing the *hidden layer output matrix*

$$\mathbf{H} = \begin{bmatrix} g(\mathbf{w}_1^T\mathbf{x}_1 + b_1) & \dots & g(\mathbf{w}_{\tilde{N}}^T\mathbf{x}_1 + b_{\tilde{N}}) \\ \vdots & \ddots & \vdots \\ g(\mathbf{w}_1^T\mathbf{x}_N + b_1) & \dots & g(\mathbf{w}_{\tilde{N}}^T\mathbf{x}_N + b_{\tilde{N}}) \end{bmatrix} \in \mathbb{R}^{N\times\tilde{N}},$$

and the matrix

$$\mathbf{B} = \begin{bmatrix} \boldsymbol{\beta}_1^T \\ \vdots \\ \boldsymbol{\beta}_{\tilde{N}}^T \end{bmatrix} \in \mathbb{R}^{\tilde{N}\times k},$$

equation (24) can be written as

$$\mathbf{T} = \mathbf{H}\mathbf{B} \tag{25}$$

for the N training samples. The goal is then to find parameters $\mathbf{w}_1, ..., \mathbf{w}_{\tilde{N}}$, $b_1, ..., b_{\tilde{N}}$ and $\mathbf{B}$ such that $\|\mathbf{T} - \mathbf{H}\mathbf{B}\|_F^2$ is minimal, i.e. to solve

$$\underset{\mathbf{w}_i, b_i, \mathbf{B}}{\arg\min} \|\mathbf{T} - \mathbf{H}\mathbf{B}\|_F^2. \tag{26}$$

This is usually done by using some gradient descent-based methods. However, as shown in [4], the weights $\mathbf{w}_i$ and bias' b_i can be randomly chosen from some continuous probability distribution, and (26) can instead be solved by finding the least squares solution (9),

$$\hat{\mathbf{B}} = \mathbf{H}^\dagger\mathbf{T}, \tag{27}$$

as long as g is infinitely differentiable.

Some common activation functions include the *sigmoid* function

$$g(x) = \frac{1}{1 + e^{-x}}, \tag{28}$$

and the *softplus* function

$$g(x) = \ln(1 + e^x). \tag{29}$$

As in Section III-C, a regularization term can be added to create a regularized extreme learning machine (RELM) [5], so that

$$\hat{\mathbf{B}} = (\mathbf{H}^T\mathbf{H} + \lambda\mathbf{I})^{-1}\mathbf{H}^T\mathbf{T}. \tag{30}$$

F. Support vector machine

Support vector machine (SVM) is a so called *maximum margin* classifier [6]. Consider a classification problem with two classes $\mathcal{C}_1$ and $\mathcal{C}_2$ whose training data $\mathbf{x}_1, ..., \mathbf{x}_N$ is linearly separable. If the training data is not linearly separable, a *feature mapping* $\boldsymbol{\phi}(\mathbf{x})$ to a higher dimensional vector space can be introduced [3]. Let the data point $\mathbf{x}_i$ be labeled with $t_i = +1$ if it belongs to $\mathcal{C}_1$, and with $t_i = -1$ if it belongs to $\mathcal{C}_2$. With the SVM algorithm one seeks to find parameters $\mathbf{w} \in \mathbb{R}^D$ and $b \in \mathbb{R}$ for a hyperplane

$$y(\mathbf{x}) = \mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}) + b \tag{31}$$

that seperates the two classes, such that the orthogonal distance from the hyperplane to the closest training samples is maximized. This distance is called the *margin* and the vectors $\mathbf{x}_i$ that lie on the margin are called *support vectors*. As shown in [6], finding the optimal hyperplane (31) is equivalent to solving the minimization problem

$$\min_{\mathbf{w}} \|\mathbf{w}\|_2^2, \tag{32}$$

under the constraints

$$t_i y(\mathbf{x}_i) \geq 1, \ i = 1, ..., N, \tag{33}$$

or equivalently, as shown in [3], by solving the dual problem

$$\max_{\mathbf{a}} \left(\sum_{i=1}^{N} a_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} a_i a_j t_i t_j k(\mathbf{x}_i, \mathbf{x}_j) \right) \tag{34}$$

under the constraints

$$a_i \geq 0, \ i = 1, ..., N \tag{35}$$

and

$$\sum_{i=1}^{N} a_i t_i = 0, \tag{36}$$

where $\mathbf{a} = (a_1, ..., a_N)^T$ and $k(\mathbf{x}, \mathbf{x}') = \boldsymbol{\phi}(\mathbf{x})^T \boldsymbol{\phi}(\mathbf{x}')$ is the kernel function. As in Section III-D, any valid kernel can be used in place of $\boldsymbol{\phi}(\mathbf{x})^T \boldsymbol{\phi}(\mathbf{x}')$. Equation (34), together with the constraints (35) and (36), is solved with numerical optimization methods to find the optimal solution $\hat{\mathbf{a}}$ [3]. The optimal vector $\hat{\mathbf{w}}$ is then determined using

$$\hat{\mathbf{w}} = \sum_{i=1}^{N} \hat{a}_i t_i \boldsymbol{\phi}(\mathbf{x}_i), \tag{37}$$

and $\hat{b}$ is determined using the constraints (33).

It may sometimes be preferable to let some data points be misclassified or lie on the wrong side of the margin when determining a hyperplane. This concept is referred to as using a *soft margin*, and is done in order to create either a larger margin or a less complex model, or both [3]. In this way unusual data points may be taken into less consideration, thus avoiding creating an overfitted model. A soft margin is implemented by adding a penalization term to the minimization problem (32), instead solving

$$\min_{\mathbf{w}} \left(\|\mathbf{w}\|_2^2 + C \sum_{i=1}^{N} \xi_i \right), \tag{38}$$

with constraints

$$t_i y(\mathbf{x}_i) \geq 1 - \xi_i, \ \xi_i \geq 0, \ i = 1, ..., N, \tag{39}$$

and

$$0 \leq a_i \leq C, \ i = 1, ..., N, \tag{40}$$

where $\xi_i = 0$ for data points $\mathbf{x}_i$ on the correct side of the margin and $\xi_i = |t_i - y(\mathbf{x}_i)|$ otherwise [3]. $C > 0$ is a hyperparameter that governs the softness of the margin since the size of C determines how many training samples are allowed to be misclassified or lie on the wrong side of the margin.

When the optimal hyperplane is found, a new input $\mathbf{x}$ can be predicted to belong to either class $\mathcal{C}_1$ or $\mathcal{C}_2$, depending on which side of the hyperplane $\boldsymbol{\phi}(\mathbf{x})$ lies, i.e. depending on the sign of $y(\mathbf{x})$.

Equations (32)-(39) are gathered from [3] and [6]. For a detailed explanation of the SVM algorithm, soft margin, and of the derivations of equations (32)-(39), consult the above sources.

G. Validating the hyperparameters

In order to find the optimal values of the hyperparameters of a given algorithm one can divide the set of training samples D_{tr} into two parts; a new training set $D_{tr,new}$ and a validation set D_{val}. The algorithm can then be trained multiple times over a range of different hyperparameter values using $D_{tr,new}$. The performance of the algorithm in each training instance is evaluated by calculating the accuracy achieved when predicting the classes of the vectors in D_{val} and the hyperparameter values that result in the highest validation accuracy are chosen. This is usually done multiple times, for multiple random shuffles of D_{tr}, resulting in a set L of values for a given hyperparameter. One then chooses the most frequently recorded value in L as the optimal value of that hyperparameter.

IV. Methodology

A. Data gathering and image preprocessing

A total of 1000 images were gathered from the Swedish clothing-retail company H&M's website [11]; 500 of boy's clothes and 500 of girl's clothes. The clothes were categorized as being for children between the ages of 18 months and 10 years. The format of the images were .jpeg and the resolutions were 768×1152.

The images were preprocessed using the openCV library [12], version $4.0.0$. With openCV, the images were converted to two-dimensional arrays with 3-channel pixel values, one value each for the colors red, green, and blue between 0 and 255. The resolution of the images was decreased to 32×48 using openCV's interpolation method INTER_AREA, as shown in Fig. 2. Thus the dimension of the input $\mathbf{x}$ was decreased to $32 \times 48 \times 3 = 4608$. This was done since the original images had a high resolution and would have required a lot of memory and computational power to work with.

The pixel values were normalized to lie between 0 and 1 by dividing each value with 255. The arrays were then vectorized using NumPy's ravel method [13].

Fig. 2. An example image before and after resizing using openCV's INTER_AREA interpolation method. Image from [11].

The data was randomly shuffled ten times and for each shuffle it was divided into two parts; a training set $D_{tr}^{(i)}$ consisting of 750 images and a testing set $D_{ts}^{(i)}$ consisting of 250 images, $i = 1, ..., 10$.

B. Training and validation

Three different algorithms were implemented using Python version 3.6.8; kernel ridge regression with the GRBF kernel (23), regularized extreme learning machine, and support vector machine with a soft margin and the GRBF kernel. KRR and RELM were implemented from the ground up as explained in Sec. III. The support vector machine was implemented using the package SVM from [14]. For RELM, the weights $\mathbf{w}_1, ..., \mathbf{w}_{\tilde{N}}$ and the bias' $b_1, ..., b_{\tilde{N}}$ for each hidden node were randomly generated from NumPy's uniform probability distribution [13] in the range $[-1, 1]$. $\tilde{N}$ was set to 5000, i.e. a little larger than the dimension of the input, and the softplus function (29) was used as activation. For SVM and KRR, the hyperparameter s for the GRBF kernel was set to the average Euclidean distance between all pairs of training samples. The two classes were represented by target vectors; $\mathbf{t} = (1, 0)^T$ for the boy class and $\mathbf{t} = (0, 1)^T$ for the girl class. The code used in the project can be seen in the Appendix.

Each training set $D_{tr}^{(i)}$, $i = 1, ..., 10$, was randomly shuffled and split into two parts as explained in Sec. III-G; 80% of $D_{tr}^{(i)}$ was used for $D_{tr,new}^{(i)}$, i.e. 600 images, and 20% was used for $D_{val}^{(i)}$, i.e. 150 images. For all training sets, KRR, RELM, and SVM were trained for all $\lambda, C \in \{10^{-10}, 10^{-9}, ..., 10^{10}\}$, where λ is the regularization constant and C the soft margin constant. The performance for each λ and C was calculated using $D_{val}^{(i)}$ and the values of λ and C that achieved the highest validation accuracy were recorded for each algorithm.

The above procedure, i.e. randomly shuffling and splitting $D_{tr}^{(i)}$ and finding the best performing hyperparameters, was done a total of ten times and the most frequently recorded values of λ as well as C were chosen. Thus, for each $D_{tr}^{(i)}$ an optimal $\hat{\lambda}_{krr}^{(i)}$, $\hat{\lambda}_{relm}^{(i)}$, and $\hat{C}^{(i)}$ was determined for KRR, RELM, and SVM, respectively, $i = 1, ..., 10$.

C. Testing

The algorithms were retrained on each $D_{tr}^{(i)}$ using the values $\hat{\lambda}_{krr}^{(i)}$, $\hat{\lambda}_{relm}^{(i)}$, and $\hat{C}^{(i)}$ that were chosen in the validation phase. The performance was evaluated on $D_{ts}^{(i)}$ by calculating the classification accuracy, thus resulting in ten accuracies $a_1, ..., a_{10}$ for each algorithm. In each case, the total accuracies of the algorithms were calculated as the average $\bar{a}$ of all testing accuracies and the standard deviation was calculated using

$$\sigma = \sqrt{\frac{\sum_{i=1}^{M_{ts}} (a_i - \bar{a})^2}{M_{ts} - 1}}, \tag{41}$$

where $M_{ts} = 10$ [2]. For RELM the above procedure was repeated for all $\tilde{N} \in \{0, 100, ..., 6000\}$ in order to see how the accuracy changes with the number of nodes.

All the training, validation, and testing was done on a computer with an Intel Core i7-6700 3.40GHz CPU, 32GB of RAM, and with an Ubuntu 16.04 operating system.

V. Results

A. Validation

Figs. 3-5 show the total validation accuracy for all algorithms as a function of the values of the hyperparameters, together with the standard deviations. The total accuracy was calculated as the average over all accuracies achieved on $D_{val}^{(i)}$, $i = 1, ..., 10$, for all splits and shuffles. We can see that the validation accuracy for each algorithm varies with the values of the hyperparameters. For KRR, the optimal λ over all validation sets and splits was $\hat{\lambda}_{krr} = 10^{-2}$, whilst for RELM the optimal λ was $\hat{\lambda}_{relm} = 10^{5}$. For SVM, the optimal value of the soft margin constant C was $\hat{C} = 10^2$.

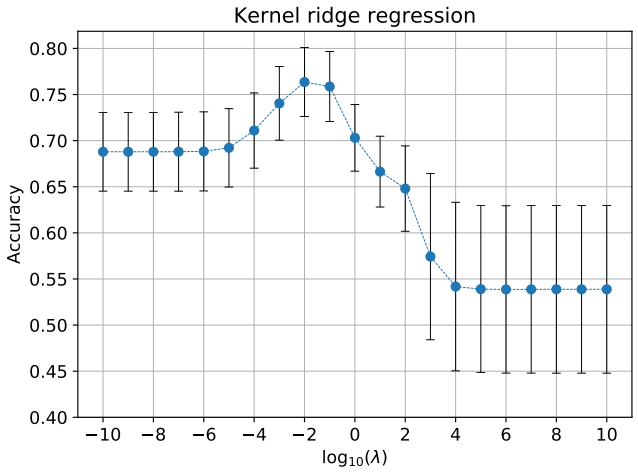

Fig. 3. Average validation accuracy for ridge regression with a GRBF kernel as a function of the logarithm of the regularization constant λ for all ten shuffles together with the standard deviation. The hyperparameter s was set to the average Euclidean distance between all pairs of training samples.

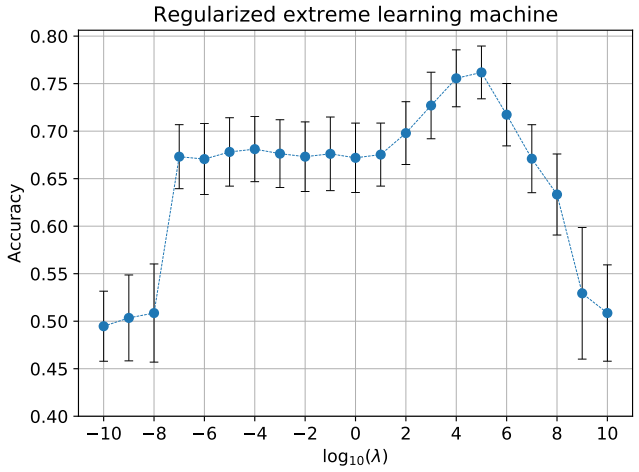

Fig. 4. Average validation accuracy for regularized extreme learning machine with $\tilde{N} = 5000$ as a function of the logarithm of the regularization constant λ for all ten shuffles together with the standard deviation.

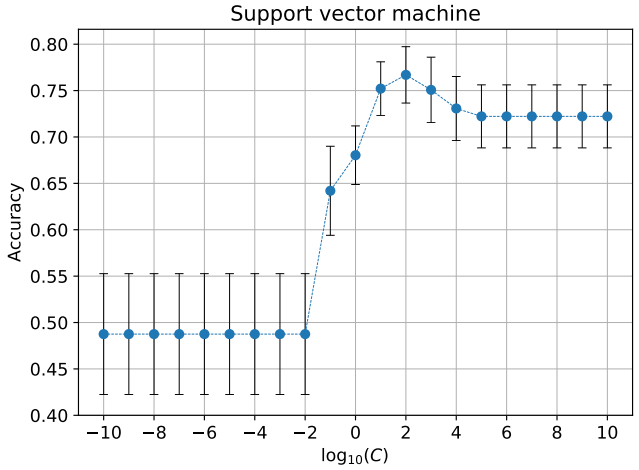

Fig. 5. Average validation accuracy for support vector machine with a GRBF kernel as a function of the logarithm of the soft margin constant C for all ten shuffles together with the standard deviation. The hyperparameter s was set to the average Euclidean distance between all pairs of training samples.

B. Testing

In Fig. 6, the relationship between the average testing accuracy for RELM on all $D_{ts}^{(i)}$, $i = 1, ..., 10$, and the number of hidden nodes is shown. We can see that the testing accuracy increases with the number of hidden nodes. However, at around 3000 hidden nodes the accuracy saturates and stays approximately constant.

Table I shows the average total testing accuracy for the algorithms as well as the average accuracies achieved on each class. The table also includes the average time spent on training and testing for each algorithm. The highest total accuracy achieved and the shortest training and testing time are highlighted in bold. We can see that SVM had the highest total testing accuracy, 76.9%, and the smallest error, 1.9%, although all three algorithms achieved roughly the same accuracy. Furthermore, SVM had the shortest training and testing time.

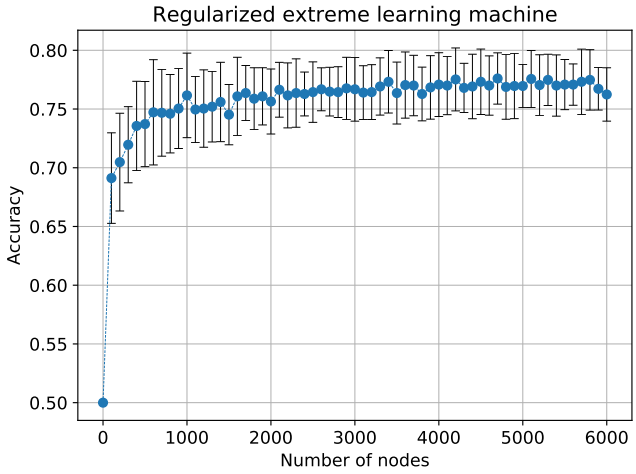

Fig. 6. Average testing accuracy for regularized extreme learning machine as a function of the number of hidden nodes $\tilde{N}$ together with the standard deviation. The optimal regularization constant $\hat{\lambda}_{relm}^{(i)}$ for each test set was determined in the validation phase.

TABLE I
AVERAGE TOTAL TESTING ACCURACY AND THE TIME ELAPSED DURING TRAINING AND TESTING FOR ALL ALGORITHMS TOGETHER WITH THE TESTING ACCURACIES FOR THE TWO CLASSES.

Algorithm	Accuracy (%)			Time (s)	
	Boy class	Girl class	Total	Training	Testing
KRR	75.8 ± 2.6	77.3 ± 3.5	76.6 ± 2.3	6.9 ± 0.2	1.7 ± 0.1
RELM	75.9 ± 3.1	77.4 ± 4.2	76.7 ± 2.6	11.2 ± 0.4	4.1 ± 0.2
SVM	76.8 ± 2.4	77.0 ± 3.9	**76.9 ± 1.9**	**4.0 ± 0.2**	**0.6 ± 0.1**

VI. Discussion

A. Validation results

In Fig. 3, we can see that the validation accuracy for KRR increases with increasing λ up to $\hat{\lambda}_{krr}$ where it reaches about 76%. It then starts to quickly decrease and for a too large value of λ, i.e. a too large penalty on the size of the model parameters, the accuracy drops to about 54%. For small values of λ, the validation accuracy for KRR is only about 68%. This indicates that the model is overfitted and thus a larger penalty on the size of the model parameters is needed. A similar behaviour can be seen for RELM in Fig. 4. However, for $\lambda \in \{10^{-10}, 10^{-9}, 10^{-8}\}$, the validation accuracy gets as low as approximately 49% for RELM, which indicates that RELM is even more overfitted than KRR for such small values.

The validation accuracy for SVM is constant at around 49% for all values of C up to $C = 10^{-2}$, as can be seen in Fig. 5. With too small values of the soft margin constant a lot of training samples are allowed to be misclassified, which leads to a poor model. The accuracy then quickly increases to about 76% for $\hat{C}$ and then stabilizes at around 72% for larger values of C. This is expected since large values of C leads to a harder margin where essentially no training samples are allowed to be misclassified or lie on the wrong side of the margin.

B. Testing results

For RELM, $\tilde{N}$ was set to 3000 during testing because of the saturation trend that can be seen in Fig. 6. A further increase of the number of nodes would only lead to a longer training and testing time and not a substantial increase in accuracy.

As can be seen in Table I, all three algorithms achieved approximately the same total testing accuracy, 77%. One of the reasons for this could be that no more information is available in the data set. However, to determine if this is the case one would need to examine the data set more closely and compare the misclassified images for each algorithm.

C. Time complexity

RELM had significantly longer training and testing times than the two other methods. Since the input dimension $D = 4608$ was rather large, a large number of hidden nodes was necessary in order to achieve an acceptable accuracy, which resulted in a large hidden layer matrix $\mathbf{H}$. During training, most time was spent on creating the matrices $\mathbf{K} \in \mathbb{R}^{750\times750}$ and $\mathbf{H} \in \mathbb{R}^{750\times3000}$, for KRR and RELM, respectively. Therefore, since the dimensions of $\mathbf{K}$ were much smaller than the dimensions of $\mathbf{H}$, KRR had a faster training time. One should note that the implementation of SVM is probably more optimized than our implementations of KRR and RELM. This could be one of the reasons why SVM is faster than both KRR and RELM.

D. Future work

To further investigate the classification accuracy that a machine can achieve on our data set, one could implement and test more kernels and activation functions. Experiments with convolutional neural networks or other more advanced algorithms could also be of interest. Furthermore, more images could be gathered to see how well the results generalizes to a larger data set. The dependence of the testing accuracy on the image resolutions could also be investigated. Since lower resolutions result in faster training and testing times, it is preferable to use as low a resolution as possible.

A study in how the testing accuracy depends on the size of the training set could also be performed. Such a study could be comparable to a child's increasing ability to correctly classify images with increasing age, since we tend to get better at classifying objects if we have seen more of them. Furthermore, in order to get a perspective on how the achieved results compare to a child's ability to classify the data, a human test study could be conducted. The test subjects could then be shown both the original images and the images with reduced resolution to see how well they perform on both.

VII. Summary and conclusions

All three of the implemented algorithms – kernel ridge regression, regularized extreme learning machine, and support vector machine – achieved similar testing results, although SVM achieved the highest accuracy of 76.9% with an error of $\pm1.9\%$. As such, we conclude that with the used algorithms, a machine is capable of classifying children's clothes by gender with an accuracy of approximately 77%. However, if more advanced algorithms were to be implemented and tested, a higher accuracy may be attained. How these results compare to a child's ability to classify clothes by gender needs further investigation, e.g. via a human study.

Appendix
Code used in the project

Acknowledgment

The authors would like to thank our supervisors, Saikat Chatterjee and Alireza M. Javid, for all their ideas, guidance, and patience during the course of this project.

References

[1] C. L. Martin and D. N. Ruble, "Patterns of gender development," *Annual Review of Psychology*, vol. 61, no. 1, pp. 353–381, Jan. 2010.

[2] T. Hastie, R. Tibshirani, and J. H. Friedman, *The elements of statistical learning: data mining, inference, and prediction*, 2nd ed. New York: Springer, 2009.

[3] C. M. Bishop, *Pattern recognition and machine learning*. New York: Springer, 2006.

[4] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew, "Extreme learning machine: Theory and applications," *Neurocomputing*, vol. 70, no. 1-3, pp. 489–501, Dec. 2006.

[5] W. Deng, Q. Zheng, and L. Chen, "Regularized extreme learning machine," in *2009 IEEE Symposium on Computational Intelligence and Data Mining*, March 2009, pp. 389–395.

[6] B. E. Boser, I. M. Guyon, and V. N. Vapnik, "A training algorithm for optimal margin classifiers," in *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*. New York: ACM, 1992, pp. 144–152.

[7] S. Cai, J. Wang, and L. Quan, "How fashion talks: Clothing-region-based gender recognition," in *Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications*. Cham, Switzerland: Springer International Publishing, 2014, pp. 515–523.

[8] B. Li, X.-C. Lian, and B.-L. Lu, "Gender classification by combining clothing, hair and facial component classifiers," *Neurocomputing*, vol. 76, no. 1, pp. 18 – 27, Jan. 2012.

[9] K. Ueki, H. Komatsu, S. Imaizumi, K. Kaneko, S. Imaizumi, N. Sekine, J. Katto, and T. Kobayashi, "A method of gender classification by integrating facial, hairstyle, and clothing images," in *Proceedings of the 17th International Conference on Pattern Recognition*, vol. 4, Cambridge, UK, Aug. 2004, pp. 446–449.

[10] K. B. Petersen and M. S. Pedersen. (2012, Nov.) The Matrix Cookbook. [Online]. Available: http://www2.imm.dtu.dk/pubdb/views/publication_details.php?id=3274

[11] (2019, Apr.) Hennes & Mauritz. [Online]. Available: http://www.hm.se

[12] (2019, Apr.) Open source computer vision library. [Online]. Available: https://opencv.org/

[13] T. Oliphant, "NumPy: A guide to NumPy," USA: Trelgol Publishing, 2006. [Online]. Available: http://www.numpy.org/

[14] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, Oct. 2011.