Fitting Penalized Logistic Regression Models using QR Factorization[⋆]

Jacek Klimaszewski[0000−0003−4885−2475] and Marcin Korzeń[0000−0002−1306−7146]

Faculty of Computer Science and Information Technology
West Pomeranian University of Technology in Szczecin, Szczecin, Poland
{jklimaszewski,mkorzen}@wi.zut.edu.pl

Abstract. The paper presents improvement of a commonly used learning algorithm for logistic regression. In the direct approach Newton method needs inversion of Hessian, what is cubic with respect to the number of attributes. We study a special case when the number of samples m is smaller than the number of attributes n, and we prove that using previously computed QR factorization of the data matrix, Hessian inversion in each step can be performed significantly faster, that is $\mathcal{O}\left(m^3\right)$ or $\mathcal{O}\left(m^2 n\right)$ instead of $\mathcal{O}\left(n^3\right)$ in the ordinary Newton optimization case. We show formally that it can be adopted very effectively to ℓ^2 penalized logistic regression and also, not so effectively but still competitively, for certain types of sparse penalty terms. This approach can be especially interesting for a large number of attributes and relatively small number of samples, what takes place in the so-called extreme learning. We present a comparison of our approach with commonly used learning tools.

Keywords: Newton Method · Logistic Regression · Regularization · QR Factorization.

1 Introduction

We consider a task of binary classification problem with n inputs and with one output. Let $\mathbf{X} \in \mathbb{R}^{m\times n}$ be a dense data matrix including m data samples and n attributes, and $\boldsymbol{y}_{m\times 1}$, $y_i \in \{-1, +1\}$ are corresponding targets. We consider the case $m < n$. In the following part bold capital letters $\mathbf{X}, \mathbf{Y}, \ldots$ denote matrices, bold lower case letters $\boldsymbol{x}, \boldsymbol{w}$ stand for vectors, and normal lower case x_{ij}, y_i, λ for scalars. The paper concerns classification, but it is clear that the presented approach can be easily adopted to the linear regression model.

We consider a common logistic regression model in the following form:

$$Pr(y = +1|\boldsymbol{x}, \boldsymbol{w}) \equiv \sigma(\boldsymbol{x}, \boldsymbol{w}) = \frac{1}{1 + e^{-\sum_{j=1}^{n} x_j w_j}}. \tag{1}$$

[⋆] This work was financed by the National Science Centre, Poland. Research project no.: 2016/21/B/ST6/01495

ICCS Camera Ready Version 2020
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-50417-5_4

Learning of this model is typically reduced to the optimization of negative log-likelihood function (with added regularization in order to improve generalization and numerical stability):

$$L(\boldsymbol{w}) = \lambda P(\boldsymbol{w}) + \sum_{i=1}^{m} \log(1 + e^{-y_i \cdot \sum_{j=1}^{n} x_{ij} w_j}), \tag{2}$$

where $\lambda > 0$ is a regularization parameter. Here we consider two separate cases:

1. rotationally invariant case, i.e. $P(\boldsymbol{w}) = \frac{1}{2}\|\boldsymbol{w}\|_2^2$,
2. other (possibly non-convex) cases, including $P(\boldsymbol{w}) = \frac{1}{q}\|\boldsymbol{w}\|_q^q$.

Most common approaches include IRLS algorithm [7, 15] and direct Newton iterations [14]. Both approaches are very similar — here we consider Newton iterations:

$$\boldsymbol{w}^{(k+1)} = \boldsymbol{w}^{(k)} - \alpha \mathbf{H}^{-1}\boldsymbol{g}, \tag{3}$$

where step size α is chosen via backtracking line search [1]. Gradient $\boldsymbol{g}$ and Hessian $\mathbf{H}$ of $L(\boldsymbol{w})$ have a form:

$$\boldsymbol{g} = \lambda \frac{\partial P}{\partial \boldsymbol{w}} + \sum_{i=1}^{m} y_i \cdot (\sigma(\boldsymbol{x}_i, \boldsymbol{w}) - 1) \cdot \boldsymbol{x}_i \equiv \lambda \frac{\partial P}{\partial \boldsymbol{w}} + \mathbf{X}^T \boldsymbol{v}, \tag{4}$$

$$\mathbf{H} = \lambda \frac{\partial^2 P}{\partial \boldsymbol{w} \partial \boldsymbol{w}^T} + \mathbf{X}^T\mathbf{D}\mathbf{X} \equiv \mathbf{E} + \mathbf{X}^T\mathbf{D}\mathbf{X}, \tag{5}$$

where $\mathbf{D}$ is a diagonal matrix, whose i-th entry equals $\sigma(\boldsymbol{x}_i, \boldsymbol{w}) \cdot (1 - \sigma(\boldsymbol{x}_i, \boldsymbol{w}))$, and $v_i = y_i \cdot (\sigma(\boldsymbol{x}_i, \boldsymbol{w}) - 1)$.

Hessian is a sum of the matrix $\mathbf{E}$ (second derivative of the penalty function multiplied by λ) and the matrix $\mathbf{X}^T\mathbf{D}\mathbf{X}$. Depending on the penalty function P, the matrix $\mathbf{E}$ may be: 1) scalar diagonal ($\lambda\mathbf{I}$), 2) non-scalar diagonal, 3) other type than diagonal. In this paper we investigate only cases 1) and 2).

Related works. There are many approaches to learning logistic regression model, among them there are direct second order procedures like IRLS, Newton (with Hessian inversion using linear conjugate gradient) and first order procedures with nonlinear conjugate gradient as the most representative example. A short review can be found in [14]. The other group of methods includes second order procedures with Hessian approximation like L-BFGS [21] or fixed Hessian, or truncated Newton [2, 13]. Some of those techniques are implemented in `scikit-learn` [17], which is the main environment for our experiments. QR factorization is a common technique of fitting the linear regression model [15, 9].

2 Procedure of optimization with QR decomposition

Here we consider two cases. The number of samples and attributes leads to different kinds of factorization:

– LQ factorization for $m < n$,
– QR factorization for $m \geqslant n$.

Since we assume $m < n$, we consider LQ factorization of matrix $\mathbf{X}$:

$$\mathbf{X} = \mathbf{LQ} = \left[\hat{\mathbf{L}}\ \mathbf{0}\right] \cdot \begin{bmatrix} \hat{\mathbf{Q}} \\ \tilde{\mathbf{Q}} \end{bmatrix} = \hat{\mathbf{L}}\hat{\mathbf{Q}}, \tag{6}$$

where $\hat{\mathbf{L}}$ is $m \times m$ lower triangular matrix, $\mathbf{Q}$ is $n \times n$ orthogonal matrix and $\hat{\mathbf{Q}}$ is $m \times n$ semi-orthogonal matrix ($\hat{\mathbf{Q}}\hat{\mathbf{Q}}^T = \mathbf{I}$, $\tilde{\mathbf{Q}}\hat{\mathbf{Q}}^T = \mathbf{0}$). The result is essentially the same as if QR factorization of the matrix $\mathbf{X}^T$ was performed.

Finding the Newton direction from the eq. (3):

$$\boldsymbol{d} = \mathbf{H}^{-1}\boldsymbol{g} \tag{7}$$

involves matrix inversion, which has complexity $\mathcal{O}(n^3)$. A direct inversion of Hessian can be replaced (and improved) with a solution of the system of linear equations:

$$\mathbf{H}\boldsymbol{d} = \boldsymbol{g}, \tag{8}$$

with the use of the conjugate gradient method. This Newton method with Hessian inversion using linear conjugate gradient is an initial point of our research. We show further how this approach can be improved using QR decomposition.

2.1 The ℓ^2 penalty case and rotational invariance

In the ℓ^2-regularized case solution has a form:

$$\boldsymbol{d} = \left(\mathbf{X}^T\mathbf{D}\mathbf{X} + \lambda\mathbf{I}\right)^{-1}\left(\mathbf{X}^T\boldsymbol{v} + \lambda\boldsymbol{w}\right). \tag{9}$$

Substituting $\mathbf{LQ}$ for $\mathbf{X}$ and $\hat{\mathbf{Q}}^T\hat{\mathbf{Q}}\boldsymbol{w}$ for $\boldsymbol{w}$:

$$\frac{\partial}{\partial \boldsymbol{w}}\left(\frac{1}{2}\cdot\|\hat{\mathbf{Q}}\boldsymbol{w}\|_2^2\right) = \hat{\mathbf{Q}}^T\hat{\mathbf{Q}}\boldsymbol{w} \tag{10}$$

in the eq. (9) leads to:

$$\begin{aligned}
\boldsymbol{d} &= \left(\mathbf{Q}^T\mathbf{L}^T\mathbf{D}\mathbf{L}\mathbf{Q} + \lambda\mathbf{I}\right)^{-1}\left(\mathbf{Q}^T\mathbf{L}^T\boldsymbol{v} + \lambda\hat{\mathbf{Q}}^T\hat{\mathbf{Q}}\boldsymbol{w}\right) \\
&= \left[\mathbf{Q}^T\left(\mathbf{L}^T\mathbf{D}\mathbf{L} + \lambda\mathbf{I}\right)\mathbf{Q}\right]^{-1}\left(\mathbf{Q}^T\mathbf{L}^T\boldsymbol{v} + \lambda\hat{\mathbf{Q}}^T\hat{\mathbf{Q}}\boldsymbol{w}\right) \\
&= \mathbf{Q}^T\left(\mathbf{L}^T\mathbf{D}\mathbf{L} + \lambda\mathbf{I}\right)^{-1}\mathbf{Q}\left(\mathbf{Q}^T\mathbf{L}^T\boldsymbol{v} + \lambda\hat{\mathbf{Q}}^T\hat{\mathbf{Q}}\boldsymbol{w}\right) \\
&= \left[\hat{\mathbf{Q}}^T\ \tilde{\mathbf{Q}}^T\right]\cdot\begin{bmatrix}\hat{\mathbf{L}}^T\mathbf{D}\hat{\mathbf{L}} + \lambda\mathbf{I} & \mathbf{0} \\ \mathbf{0} & \lambda\mathbf{I}\end{bmatrix}^{-1}\cdot\left(\begin{bmatrix}\hat{\mathbf{L}}^T \\ \mathbf{0}\end{bmatrix}\cdot\boldsymbol{v} + \begin{bmatrix}\lambda\hat{\mathbf{Q}}\boldsymbol{w} \\ \mathbf{0}\end{bmatrix}\right) \\
&= \hat{\mathbf{Q}}^T\left(\hat{\mathbf{L}}^T\mathbf{D}\hat{\mathbf{L}} + \lambda\mathbf{I}\right)^{-1}\left(\hat{\mathbf{L}}^T\boldsymbol{v} + \lambda\hat{\mathbf{Q}}\boldsymbol{w}\right).
\end{aligned} \tag{11}$$

Algorithm 1 Newton method for ℓ^2 penalized Logistic Regression with QR factorization using transformation into a smaller space (`L2-QR`).

```
Input: $\mathbf{X} = \mathbb{R}^{m\times n}$, $\boldsymbol{y}_{m\times 1}$, $m < n$
Initialization: $[\hat{\mathbf{L}}, \hat{\mathbf{Q}}] = \mathrm{lq}(\mathbf{X})$, $\hat{\boldsymbol{w}}_{m\times 1} = \mathbf{0}$
repeat
    Compute $\hat{\boldsymbol{g}}$ and $\mathbf{D}$ for $\hat{\boldsymbol{w}}^{(k)}$.
    Solve $\left(\hat{\mathbf{L}}^T\mathbf{D}\hat{\mathbf{L}} + \lambda\mathbf{I}\right)\cdot\hat{\boldsymbol{d}} = \hat{\boldsymbol{g}}$.
    $\hat{\boldsymbol{w}}^{(k+1)} = \hat{\boldsymbol{w}}^{(k)} - \hat{\boldsymbol{d}}\cdot\arg\min_\alpha L\left(\hat{\boldsymbol{w}}^{(k)} - \alpha\hat{\boldsymbol{d}}\right)$.
until $\|\hat{\boldsymbol{g}}\|_2^2 < \epsilon$
Output: $\boldsymbol{w} = \hat{\mathbf{Q}}^T\hat{\boldsymbol{w}}$
```

First, multiplication by $\hat{\mathbf{Q}}$ transforms $\boldsymbol{w}$ to the smaller space, then inversion is done in that space and finally, multiplication by $\hat{\mathbf{Q}}^T$ brings solution back to the original space. However, all computation may be done in the smaller space (using $\hat{\mathbf{L}}$ instead of $\mathbf{X}$ in the eq. (9)) and only final solution is brought back to the original space — this approach is summarized in the Alg. 1. In the experimental part this approach is called `L2-QR`.

This approach is not new [8, 16], however the use of this trick does not seem to be common in machine learning tools.

2.2 Rotational variance

In the case of penalty functions whose Hessian $\mathbf{E}$ is a non-scalar diagonal matrix, it is still possible to construct algorithm, which solves smaller problem via QR factorization.

Consider again (5), (6) and (7):

$$\begin{aligned}\boldsymbol{d} &= \left(\mathbf{Q}^T\mathbf{L}^T\mathbf{D}\mathbf{L}\mathbf{Q} + \mathbf{E}\right)^{-1}\boldsymbol{g}\\ &= \left[\mathbf{Q}^T\left(\mathbf{L}^T\mathbf{D}\mathbf{L} + \mathbf{Q}\mathbf{E}\mathbf{Q}^T\right)\mathbf{Q}\right]^{-1}\boldsymbol{g}\\ &= \mathbf{Q}^T\left(\mathbf{L}^T\mathbf{D}\mathbf{L} + \mathbf{Q}\mathbf{E}\mathbf{Q}^T\right)^{-1}\mathbf{Q}\boldsymbol{g}.\end{aligned} \tag{12}$$

Let $\mathbf{A} = \mathbf{Q}\mathbf{E}\mathbf{Q}^T$, $\mathbf{B} = \mathbf{L}^T\mathbf{D}\mathbf{L}$, so $\mathbf{A}^{-1} = \mathbf{Q}\mathbf{E}^{-1}\mathbf{Q}^T$. Using Shermann-Morrison-Woodbury formula [5] we may write:

$$(\mathbf{A} + \mathbf{B})^{-1} = \mathbf{A}^{-1} - \mathbf{A}^{-1}\left(\mathbf{I} + \mathbf{B}\mathbf{A}^{-1}\right)^{-1}\mathbf{B}\mathbf{A}^{-1}. \tag{13}$$

Let $\mathbf{C} = \mathbf{I} + \mathbf{B}\mathbf{A}^{-1}$. Exploiting the structure of the matrices $\mathbf{L}$ and $\mathbf{Q}$ (6) yields:

$$\begin{aligned}\mathbf{C}^{-1} &= \left(\mathbf{I} + \begin{bmatrix}\hat{\mathbf{L}}^T\mathbf{D}\hat{\mathbf{L}} & \mathbf{0}\\ \mathbf{0} & \mathbf{0}\end{bmatrix}\cdot\begin{bmatrix}\hat{\mathbf{Q}}\mathbf{E}^{-1}\hat{\mathbf{Q}}^T & \hat{\mathbf{Q}}\mathbf{E}^{-1}\tilde{\mathbf{Q}}^T\\ \tilde{\mathbf{Q}}\mathbf{E}^{-1}\hat{\mathbf{Q}}^T & \tilde{\mathbf{Q}}\mathbf{E}^{-1}\tilde{\mathbf{Q}}^T\end{bmatrix}\right)^{-1}\\ &= \begin{bmatrix}\hat{\mathbf{L}}^T\mathbf{D}\mathbf{X}\mathbf{E}^{-1}\hat{\mathbf{Q}}^T + \mathbf{I} & \hat{\mathbf{L}}^T\mathbf{D}\mathbf{X}\mathbf{E}^{-1}\tilde{\mathbf{Q}}^T\\ \mathbf{0} & \mathbf{I}\end{bmatrix}^{-1}\\ &= \begin{bmatrix}\mathbf{C}_1 & \mathbf{C}_2\\ \mathbf{0} & \mathbf{I}\end{bmatrix}^{-1} = \begin{bmatrix}\mathbf{C}_1^{-1} & -\mathbf{C}_1^{-1}\mathbf{C}_2\\ \mathbf{0} & \mathbf{I}\end{bmatrix}.\end{aligned} \tag{14}$$

Algorithm 2 Newton method for Logistic Regression with QR factorization and general regularizer.

```
Input: $\mathbf{X} = \mathbb{R}^{m \times n},\ \boldsymbol{y}_{m \times 1},\ m < n$
Initialization: $[\hat{\mathbf{L}}, \hat{\mathbf{Q}}] = \mathrm{lq}\,(\mathbf{X})$
repeat
    Compute $\boldsymbol{g}$, $\mathbf{E}$ and $\mathbf{D}$ for $\boldsymbol{w}^{(k)}$.
    Compute $\mathbf{C}_1^{-1}$.
    Compute $\boldsymbol{d}$ according to eq. (15) or eq. (20)
    $\boldsymbol{w}^{(k+1)} = \boldsymbol{w}^{(k)} - \boldsymbol{d} \cdot \arg\min_\alpha L\left(\boldsymbol{w}^{(k)} - \alpha \boldsymbol{d}\right)$.
until $\|\boldsymbol{g}\|_2^2 < \epsilon$
Output: $\boldsymbol{w}$
```

Hence only matrix $\mathbf{C}_1 = \hat{\mathbf{L}}^T \mathbf{D} \mathbf{X} \mathbf{E}^{-1} \hat{\mathbf{Q}}^T + \mathbf{I}$ of the size $m \times m$ needs to be inverted — inversion of the diagonal matrix $\mathbf{E}$ is trivial. Putting (14) and (13) into (12) and simplifying obtained expression results in:

$$\boldsymbol{d} = \left(\mathbf{E}^{-1} - \mathbf{E}^{-1} \hat{\mathbf{Q}}^T \mathbf{C}_1^{-1} \hat{\mathbf{L}}^T \mathbf{D} \mathbf{X} \mathbf{E}^{-1}\right) \boldsymbol{g}. \tag{15}$$

This approach is summarized in the Alg. 2.

Application to the smooth ℓ^1 approximation Every convex twice continuously differentiable regularizer can be put in place of ridge penalty and above procedure may be used to optimize such a problem. In this article we focused on the smoothly approximated ℓ^1-norm [12] via integral of hyperbolic tangent function:

$$\|\boldsymbol{x}\|_{1\mathrm{soft}} = \sum_{j=1}^{n} \frac{1}{a} \log\left(\cosh\left(a x_j\right)\right), \ a \geqslant 1, \tag{16}$$

and we call this model `L1-QR-soft`. In this case

$$\mathbf{E} = \mathrm{diag}\{\lambda a \left(1 - \tanh^2\left(a w_1\right)\right), \ldots, \lambda a \left(1 - \tanh^2\left(a w_n\right)\right)\}.$$

Application to the strict ℓ^1 penalty Fan and Li proposed a unified algorithm for the minimization problem (2) via local quadratic approximations [3]. Here we use the idea presented by Krishnapuram [11], in which the following inequality is used:

$$\|\boldsymbol{w}\|_1 \leq \frac{1}{2} \sum_{j=1}^{n} \left(\frac{w_j^2}{|w_j'|} + |w_j'| \right), \tag{17}$$

what is true for any $\boldsymbol{w}'$ and equality holds if and only if $\boldsymbol{w}' = \boldsymbol{w}$.

Cost function has a form:

$$L(\boldsymbol{w}) = \sum_{i=1}^{m} \log(1 + e^{-y_i \cdot \sum_{j=1}^{n} x_{ij} w_j}) + \frac{\lambda}{2} \sum_{j=1}^{n} \left(\frac{w_j^2}{|w_j'|} + |w_j'| \right). \tag{18}$$

If we differentiate penalty term, we get:

$$\frac{\lambda}{2}\frac{\partial P}{\partial \boldsymbol{w}} = \frac{\partial}{\partial \boldsymbol{w}}\left(\frac{\lambda}{2}\sum_{j=1}^{n}\frac{w_j^2}{|w'_j|} + |w'_j|\right) = \mathbf{E}\boldsymbol{w}, \tag{19}$$

where

$$\mathbf{E} = \operatorname{diag}\left\{\frac{\lambda}{|w'_1|},\ldots,\frac{\lambda}{|w'_n|}\right\} = \lambda\frac{\partial^2 P}{\partial\boldsymbol{w}\partial\boldsymbol{w}^T}.$$

Initial $\boldsymbol{w}$ must be non zero (we set it to $\mathbf{1}$), otherwise there is no progress. If $|w_j|$ falls below machine precision, we set it to zero.

Applying the idea of the QR factorization leads to the following result:

$$\begin{aligned}\boldsymbol{d} &= \left(\mathbf{E}^{-1} - \mathbf{E}^{-1}\hat{\mathbf{Q}}^T\mathbf{C}_1^{-1}\hat{\mathbf{L}}^T\mathbf{D}\mathbf{X}\mathbf{E}^{-1}\right)\left(\mathbf{X}^T\boldsymbol{v} + \mathbf{E}\boldsymbol{w}\right)\\ &= \left(\mathbf{I} - \mathbf{E}^{-1}\hat{\mathbf{Q}}^T\mathbf{C}_1^{-1}\hat{\mathbf{L}}^T\mathbf{D}\mathbf{X}\right)\cdot\left(\mathbf{E}^{-1}\mathbf{X}^T\boldsymbol{v} + \boldsymbol{w}\right).\end{aligned} \tag{20}$$

One can note that when $\boldsymbol{w}$ is sparse, corresponding diagonal elements are 0. To avoid unneccessary multiplications by zero, we rewrite product $\mathbf{X}\mathbf{E}^{-1}\hat{\mathbf{Q}}^T$ as a sum of outer products:

$$\mathbf{X}\mathbf{E}^{-1}\hat{\mathbf{Q}}^T = \sum_{j=1}^{n} e_{jj}^{-1}\hat{\boldsymbol{x}}_j \otimes \hat{\boldsymbol{q}}_j, \tag{21}$$

where $\hat{\boldsymbol{x}}_j$ and $\hat{\boldsymbol{q}}_j$ are j-th columns of matrices $\mathbf{X}$ and $\hat{\mathbf{Q}}$ respectively. Similar concept is used when multiplying matrix $\mathbf{E}^{-1}\hat{\mathbf{Q}}^T$ by a vector e.g. $\boldsymbol{z}$: j-th element of the result equals $e_{jj}^{-1}\hat{\boldsymbol{q}}_j \cdot \boldsymbol{z}$. We refer to this model as `L1-QR`.

After obtaining direction $\boldsymbol{d}$ we use backtracking line search[1] with sufficient decrease condition given by Tseng and Yun [19] with one exception: if a unit step is already decent, we seek for a bigger step to ensure faster convergence.

Application to the $\ell^{q<1}$ penalty The idea described above can be directly applied to the $\ell^{q<1}$ "norms" [10] and we call it `Lq-QR`. Cost function has a form:

$$L(\boldsymbol{w}) = \sum_{i=1}^{m}\log(1 + e^{-y_i\cdot\sum_{j=1}^{n}x_{ij}w_j}) + \frac{\lambda}{2}\sum_{j=1}^{n}\left(\frac{qw_j^2}{|w'_j|^{2-q}} + (2-q)\,|w'_j|^q\right), \tag{22}$$

where

$$\mathbf{E}^{-1} = \operatorname{diag}\left\{\frac{|w_1|^{2-q}}{\lambda q},\ldots,\frac{|w_n|^{2-q}}{\lambda q}\right\}.$$

3 Complexity of proposed methods

Cost of each iteration in the ordinary Newton method for logistic regression equals $k\cdot\left(2n^2 + n\right)$, where k is the number of conjugate gradient iterations. In general $k \le n$, so in the worst case its complexity is $\mathcal{O}\left(n^3\right)$.

[1] In the line search procedure we minimize (2) with $P(\boldsymbol{w}) = \|\boldsymbol{w}\|_1$.

Rotationally invariant case QR factorization is done once and its complexity is $\mathcal{O}\left(2m^2 \cdot \left(n - \frac{m}{3}\right)\right) = \mathcal{O}\left(m^2 n\right)$. Using data transformed to the smaller space, each step of the Newton procedure is much cheaper and it requires about km^2 operations (cost of solving system of linear equations using conjugate gradient, $k \leq m$), what is $\mathcal{O}\left(m^3\right)$ in general.

As it is shown in the experimental part, this approach dominates other optimization methods (especially exact second order procedures). Looking at the above estimations, it is clear that the presented approach is especially attractive when $m \ll n$.

Rotationally variant case In the second case the most dominating operation comes from computation of the matrix $\mathbf{C}_1$ in the eq. (15). Due to dimensionality of matrices: $\hat{\mathbf{L}}_{m\times m}, \mathbf{X}_{m\times n}$ and $\hat{\mathbf{Q}}_{m\times n}$, the complexity of computation $\mathbf{C}_1$ is $\mathcal{O}(m^2 n)$ — cost of inversion of the matrix $\mathbf{C}_1$ is less important i.e. $\mathcal{O}(m^3)$. In the case of ℓ^1 penalty taking sparsity of $\boldsymbol{w}$ into account reduces this complexity to $\mathcal{O}(m^2 \cdot \#\text{nnz})$, where #nnz is the number of non-zero coefficients.

Therefore theoretical upper bound on iteration for logistic regression with rotationally variant penalty function is $\mathcal{O}\left(m^2 n\right)$, what is better than direct Newton approach. However, looking at (15), we see that the number of multiplications is large, thus a constant factor in this estimation is large.

4 Experimental Results

In the experimental part we present two cases: 1) learning ordinary logistic regression model, and 2) learning a 2-layer neural network via extreme learning paradigm. We use following datasets:

1. Artificial dataset with 100 informative attributes and 1000 redundant attributes, informative part was produced by function `make_classification` from package `scikit-learn` and whole set was transformed introducing correlations.
2. Two micro-array datasets: leukemia [6], prostate cancer [18].
3. Artificial non-linearly separable datasets: `chessboard` 3×3 and 4×4, and `two spirals` — used for learning neural network.

As a reference we use solvers that are available in the package `scikit-learn` for `LogisticRegression` model i.e. for ℓ^2 penalty we use: `LibLinear` [4] in two variants (primal and dual), `L-BFGS`, `L2-NEWTON-CG`; For sparse penalty functions we compare our solutions with two solvers available in the `scikit-learn`: `LibLinear` and `SAGA`.

For the case ℓ^2 penalty we provide algorithm `L2-QR` presented in the section 2.1. In the “sparse” case we compare three algorithms presented in the section 2.2: `L1-QR-soft`, `L1-QR` and `Lq-QR`. Our approach `L2-QR` (Alg. 1) is computationally equivalent to the `L2-NEWTON-CG` meaning that we solve an identical optimization problem (though in the smaller space). In the case of ℓ^2 penalty all models

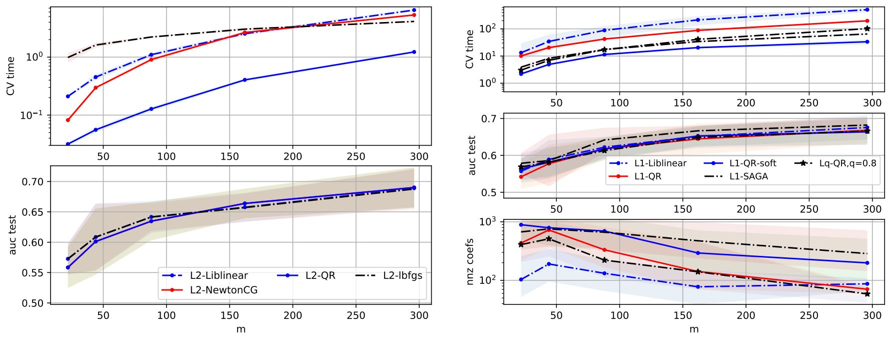

Fig. 1. Comparison of algorithms for learning ℓ^2 (a) and sparse (b) penalized logistic regressions on the artificial (300×1100) dataset. Plots present time of cross-validation procedure (`CV time`), AUC on test set (`auc test`), and number of non-zero coefficients for sparse models (`nnz coefs`).

should converge theoretically to the same solution, so differences in the final value of the objective function are caused by numerical issues (like numerical errors, approximations or exceeding the number of iterations without convergence). These differences affect the predictions on a test set.

The case of ℓ^1 penalty is more complicated to compare. The `L1-QR` Algorithm is equivalent to the `L1-Liblinear` i.e. it minimizes the same cost function. Algorithm `L1-QR-soft` uses approximated ℓ^1-norm, and algorithm `Lq-QR` uses a bit different non-convex cost function which gives similar results to ℓ^1 penalized regression for $q \approx 1$. We also should emphasize that `SAGA` algorithm does not optimize directly penalized log-likelihood function on the training set, but it is stochastic optimizer and it gives sometimes qualitatively different models. In the case `L1-QR-soft` final solution is sparse only approximately (and depends on a (16)), whereas other models produce strictly sparse models. The measure of sparsity is the number of non-zero coefficients. For `L1-QR-soft` we check the sparsity with a tolerance of order 10^{-5}.

All algorithms were started with the same parameters: maximum number of iterations (1000) and tolerance ($\epsilon = 10^{-6}$), and used the same learning and testing datasets. All algorithms depend on the regularization parameter C (or $1/\lambda$). This parameter is selected in the cross-validation procedure from the same range. During experiments with artificial data we change the size of training subset. Experiments were performed on Intel Xeoen E5-2699v4 machine, in the one threaded envirovement (with parameters `n_jobs=1` and `MKL_NUM_THREADS=1`).

Learning ordinary logistic regression model In the first experiment, presented in the Fig. 1, we use an artificial highly correlated dataset (1). We used training/testing procedure for each size of learning data, and for each classifier we select optimal value of parameter $C = 1/\lambda$ using cross-validation. The number of samples varies from 20 to 300. As we can see, in the case ℓ^2 penalty our solu-

Table 1. Experimental results for micro-array datasets and ℓ^2 penalized logistic regressions. All solvers converge to the same solution, there are only differences in times.

		$TIME_{FIT}$[S]	COST FCN.	AUC_{TEST}	ACC_{TEST}
DATASET	CLASSIFIER				
GOLUB	L2-NEWTON-CG	0.0520	1.17E+11	0.8571	0.8824
(38×7129)	L2-QR	0.0065	1.17E+11	0.8571	0.8824
	SAG	1.2560	1.17E+11	0.8571	0.8824
	LIBLINEAR L2	0.0280	1.17E+11	0.8571	0.8824
	LIBLINEAR L2 DUAL	0.0737	1.17E+11	0.8571	0.8824
	L-BFGS	0.0341	1.17E+11	0.8571	0.8824
SINGH	L2-NEWTON-CG	0.6038	5.14E+11	0.9735	0.9706
(102×12600)	L2-QR	0.0418	5.14E+11	0.9735	0.9706
	SAG	5.2822	5.13E+11	0.9735	0.9706
	LIBLINEAR L2	0.1991	5.14E+11	0.9735	0.9706
	LIBLINEAR L2 DUAL	0.6083	5.14E+11	0.9735	0.9706
	L-BFGS	0.1192	5.14E+11	0.9735	0.9706

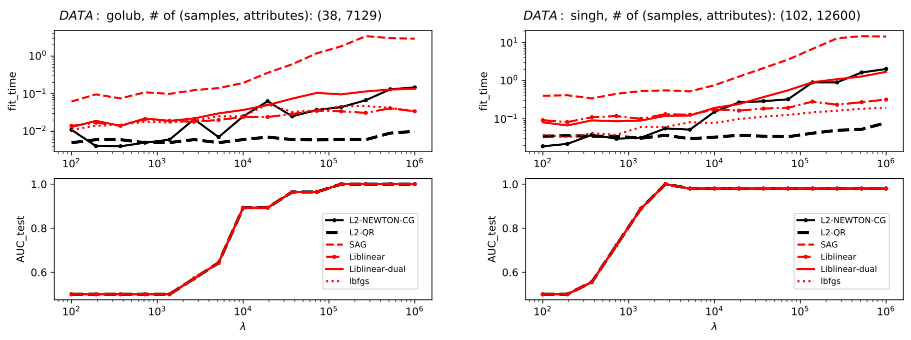

Fig. 2. Comparison of algorithms learning ℓ^2 penalized logistic regression on micro-array datasets for a sequence of λs; mean values are presented in the Tab. 1.

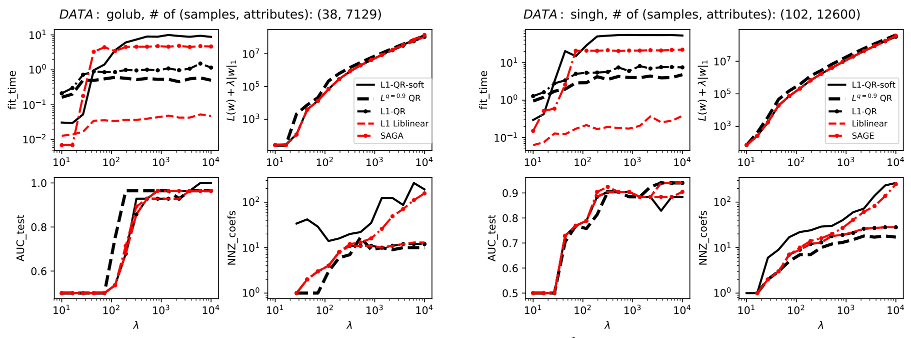

Fig. 3. Detailed comparison of algorithms learning ℓ^1 penalized logistic regression on micro-array datasets for a sequence of λs. Mean values for this case are presented in the Tab. 2.

Table 2. Experimental results for micro-array datasets and ℓ^1 penalized logistic regressions. `L1-QR` solver converges to the same solution as `L1-Liblinear`, there are only difference in times. `SAGA` and `L1-QR-soft` gives different solution.

DATASET	CLASSIFIER	$TIME_{FIT}$[s]	COST FCN.	AUC_{TEST}	ACC_{TEST}	NNZ COEFS.
GOLUB	L1-QR-SOFT	8.121	2.74E+07	0.8929	0.9118	90.1
	$L^{q=0.9}$-QR	0.544	2.80E+07	0.9393	0.95	9.1
	L1-QR	1.062	2.28E+07	0.8679	0.8912	10.2
	LIBLINEAR	0.042	2.28E+07	0.8679	0.8912	10.4
	SAGA	4.532	2.78E+07	0.8857	0.9059	46.7
SINGH	L1-QR-SOFT	51.042	6.74E+07	0.8753	0.8794	91.2
	$L^{q=0.9}$-QR	3.941	8.65E+07	0.8893	0.9	13.4
	L1-QR	6.716	6.52E+07	0.8976	0.8912	20.1
	LIBLINEAR	0.225	6.52E+07	0.8976	0.8912	20.2
	SAGA	21.251	7.11E+07	0.8869	0.8912	65.9

tion using QR decomposition `L2-QR` gives better times of fitting than ordinary solvers available in the `scikit-learn` and all algorithms work nearly the same, only `L2-lbfgs` gives slightly different results. In the case of sparse penalty our algorithm `L1-QR` works faster than `L1-Liblinear` and obtains comparable but not identical results. For sparse case `L1-SAGA` gives best predictions (about 1-2% better than other sparse algorithms), but it produces the most dense solutions similarly like `L1-QR-soft`.

In the second experiment we used micro-array data with an original train and test sets. For those datasets quotients (samples/attributes) are fixed (about 0.005–0.01). The results are shown in Tab. 1 (ℓ^2 case) and in Tab. 2 (ℓ^1 case). Tables present mean values of times and cost functions, averaged over λs. Whole traces over λs are presented in the Fig. 2 and Fig. 3. For the case of ℓ^2 penalty we notice that all tested algorithms give identical results looking at the quality of prediction and the cost function. However, time of fitting differs and the best algorithm is that, which uses QR factorization.

For the case of sparse penalty functions only algorithms `L1-Liblinear` and `L1-QR` give quantitatively the same results, however `L1-Liblinear` works about ten times faster. Other models give qualitatively different results. Algorithm `Lq-OR` obtained the best sparsity and the best accuracy in prediction and was also slightly faster than `L1-QR`. Looking at the cost function with ℓ^1 penalty we see that `L1-Liblinear` and `L1-QR` are the same, `SAGA` obtains worse cost function than even `L1-QR-soft`. We want to stress that `scikit-learn` provides only solvers for ℓ^2 and ℓ^1 penalty, not for general case ℓ^q.

Application to extreme learning and RVFL networks Random Vector Functional-link (RVFL) network is a method of learning two (or more) layer neural networks in two separate steps. In the first step coefficients for hidden neurons are chosen randomly and are fixed, and then in the second step learning algorithm is used only for the output layer. The second step is equivalent to learning the logistic regression model (a linear model with the sigmoid output function). Recently, this approach is also known as “extreme learning” (see: [20] for more references).

The output of neural network with a single hidden layer is given by:

$$y(\boldsymbol{x}; \mathbf{W}^1, \boldsymbol{b}^1, \boldsymbol{w}^2, b^2) = \varphi\Big(\sum_{j=1}^{Z} w_i^{(2)} \varphi(\boldsymbol{x}; \mathbf{W}^1(:,j), \boldsymbol{b}^1(j)) + b^2\Big), \qquad (23)$$

where: Z is the number of hidden neurons, $\varphi(\boldsymbol{x}; \boldsymbol{w}, b) = \tanh\big(\sum_{k=1}^{n} w_k x_k + b\big)$ is the activation function.

In this experiment we choose randomly hidden layer coefficients $\mathbf{W}^1$ and $\boldsymbol{b}^1$, with number of hidden neurons $Z = 1000$ and next we learn the coefficients of the output layer: $\boldsymbol{w}^2$ and b^2 using the new transformed data matrix:

$$\Phi_{m\times Z} = \varphi\big(\mathbf{X}(i,:); \mathbf{W}^1(j,:), \boldsymbol{b}^1(j)\big).$$

For experiments we prepared the class `ExtremeClassier` (in `scikit-learn` paradigm) which depends on the number of hidden neurons Z, the kind of linear output classifier and its parameters. In the fitting part we ensure the same random part of classifier. In this experiment we also added a new model — multi-layer perceptron with two layers and with Z hidden neurons fitted in the standard way using L-BFGS algorithm (`MLP-lbfgs`).

Results of the experiment are presented in the Fig. 4. For each size of learning data and for each classifier we select optimal value of parameter $C = 1/\lambda$ using cross-validation. The number of samples varies from 20 to 300. As we can see, in both cases (ℓ^2 and sparse penalties) our solution using QR decomposition gives always better times of fitting than ordinary solvers available in the `scikit-learn`. Time of fitting of `L1-QR` is 2–5 times shorter than `L1-Liblinear`, especially for the case chessboard 4×4 and two spirals. Looking at quality we see that sparse models are similar, but slightly different. For two spirals the best one is `Lq-QR` and it is also the sparsest model. Generally sparse models are better for two spirals and chessboard 4×4. The MLP model has the worst quality and comparable time of fitting to sparse regressions.

The experiment shows that use of QR factorization can effectively implement learning of RVFL network with different regularization terms. Moreover, we confirm that such learning works more stable than ordinary neural network learning algorithms, especially for the large number of hidden neurons. Exemplary decision boundaries, sparsity and found hidden neurons are shown in the Fig. 5.

5 Conclusion

In this paper we presented application of the QR matrix factorization to improve the Newton procedure for learning logistic regression models with different kind of penalties. We presented two approaches: rotationally invariant case with ℓ^2

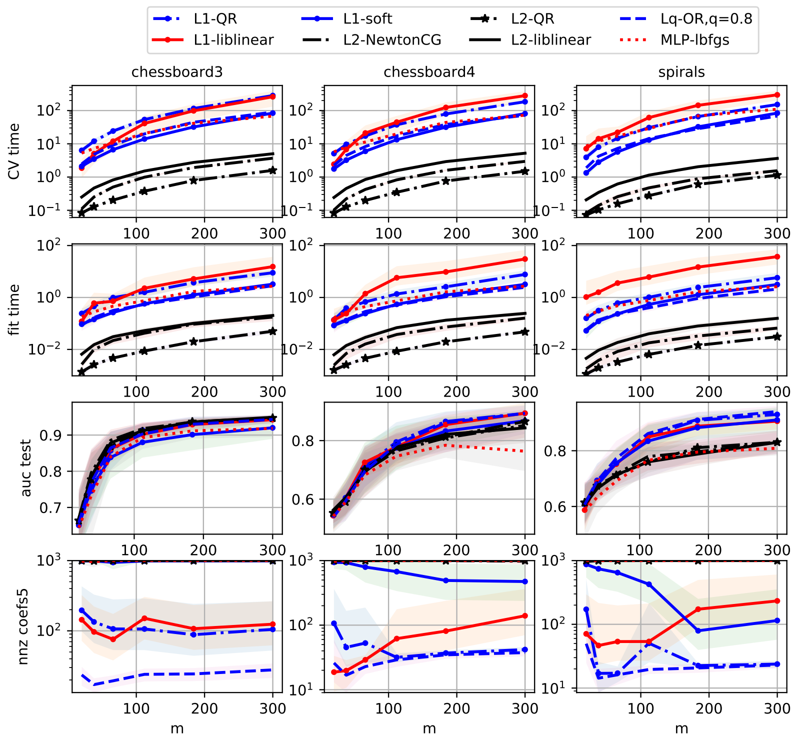

Fig. 4. Experimental results for the extreme learning. Comparison on artificial datasets. CV time is the time of cross-validation procedure, fit time is the time of fitting for the best λ, auc test is the area under ROC on test dataset, and nnz coefs5 is the number of non-zero coefficients.

penalty, and general convex rotationally variant case with sparse penalty functions. Generally speaking, there is a strong evidence that use of QR factorization in the rotational invariant case can improve classical Newton-CG algorithm when $m < n$. The most expensive operation in this approach is QR factorization itself, which is performed once at the beginning. Our experiments showed also that this approach, for $m \ll n$ surpasses also other algorithms approximating Hessian like L-BFGS and truncated Newton method (used in Liblinear). In this case we have shown that theoretical upper bound on cost of Newton iteration is $\mathcal{O}\left(m^3\right)$.

We showed also that using QR decomposition and Shermann-Morrison-Woodbury formula we can solve a problem of learning the regression model with different sparse penalty functions. Actually, improvement in this case is not as strong as in the case of ℓ^2 penalty, however we proved that using QR factorization we obtain theoretical upper bound significantly better than for general Newton-CG procedure. In fact, the Newton iterations in this case have the same cost as the initial cost of the QR decomposition i.e. $\mathcal{O}\left(m^2 n\right)$. Numerical experiments revealed that for more difficult and correlated data (e.g. for extreme

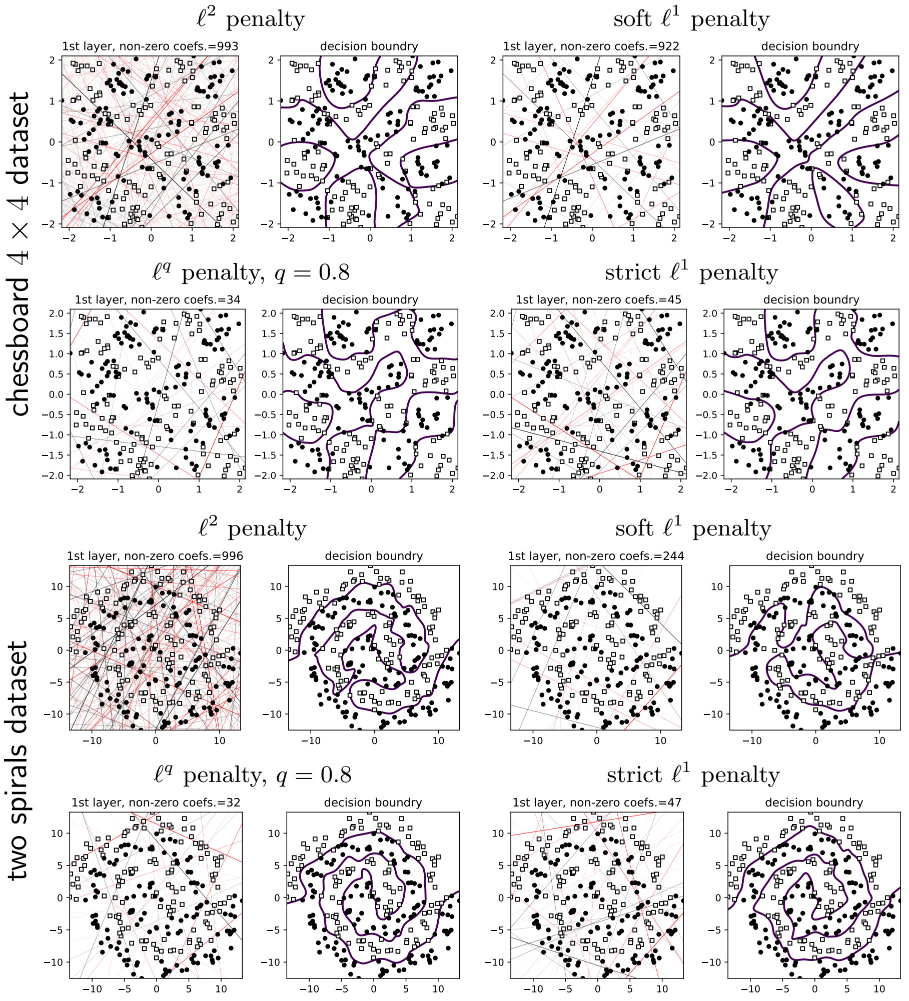

Fig. 5. Exemplary decision boundaries for different penalty functions (ℓ^2, ℓ^1 with a smooth approximation of the absolute value function, $\ell^{q=0.8}$, ℓ^1) on used datasets. In the figure there are coefficients of the first layer of the neural network represented as lines — intensity and color represents magnitude and sign of the particular coefficient.

learning) such approach may work faster than L1-Liblinear. However, we should admit that in a typical and simpler cases L1-Liblinear may be faster.

References

1. Boyd, S., Vandenberghe, L.: Convex Optimization. Cambridge University Press, New York, NY, USA (2004)
2. Dai, Y.H.: On the Nonmonotone Line Search. Journal of Optimization Theory and Applications **112**(2), 315–330 (Feb 2002)

3. Fan, J., Li, R.: Variable Selection via Nonconcave Penalized Likelihood and its Oracle Properties. Journal of the American Statistical Association **96**(456), 1348–1360 (2001)
4. Fan, R.E., Chang, K.W., Hsieh, C.J., Wang, X.R., Lin, C.J.: Liblinear: A library for large linear classification. J. Mach. Learn. Res. **9**, 1871–1874 (Jun 2008)
5. Golub, G., Van Loan, C.: Matrix Computations. Johns Hopkins Studies in the Mathematical Sciences, Johns Hopkins University Press (2013)
6. Golub, T.R., Slonim, D.K., Tamayo, P., Huard, C., Gaasenbeek, M., P., M.J., Coller, H., Loh, M.L., Downing, J.R., Caligiuri, M.A., Bloomfield, C.D., Lander, E.S.: Molecular classification of cancer: Class discovery and class prediction by gene expression monitoring. Science **286**(5439), 531–537 (1999)
7. Green, P.J.: Iteratively reweighted least squares for maximum likelihood estimation, and some robust and resistant alternatives (with discussion). Journal of the Royal Statistical Society, Series B, Methodological **46**, 149–192 (1984)
8. Hastie, T., Tibshirani, R.: Expression arrays and the $p \gg n$ problem (2003)
9. Hastie, T., Tibshirani, R., Friedman, J.: The Elements of Statistical Learning. Springer Series in Statistics, Springer New York Inc., New York, NY, USA (2001)
10. Kabán, A., Durrant, R.J.: Learning with $L_{q<1}$ vs L_1-Norm Regularisation with Exponentially Many Irrelevant Features. In: Daelemans, W., Goethals, B., Morik, K. (eds.) Machine Learning and Knowledge Discovery in Databases. pp. 580–596. Springer Berlin Heidelberg, Berlin, Heidelberg (2008)
11. Krishnapuram, B., Carin, L., Figueiredo, M.A.T., Hartemink, A.: Sparse Multinomial Logistic Regression: Fast Algorithms and Generalization Bounds. IEEE Transactions on Pattern Analysis and Machine Intelligence **27**(6), 957–968 (June 2005)
12. Lee, Y.J., Mangasarian, O.: SSVM: A Smooth Support Vector Machine for Classification. Computational Optimization and Applications **20**, 5–22 (2001)
13. Lin, C.J., Weng, R.C., Keerthi, S.S.: Trust Region Newton Method for Logistic Regression. Journal of Machine Learning Research **9**, 627–650 (2008)
14. Minka, T.P.: A comparison of numerical optimizers for logistic regression (2003), https://tminka.github.io/papers/logreg/minka-logreg.pdf
15. Murphy, K.P.: Machine learning: a probabilistic perspective. MIT Press, Cambridge, Mass. [u.a.] (2013)
16. Ng, A.Y.: Feature Selection, L_1 vs. L_2 Regularization, and Rotational Invariance. In: Proceedings of the Twenty-first International Conference on Machine Learning. pp. 78–85. ICML '04, ACM, New York, NY, USA (2004)
17. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: Machine learning in Python. Journal of Machine Learning Research **12**, 2825–2830 (2011)
18. Singh, S., Skanda, S., Scott, S., Arie, B., Sujata, P., Gurmit, S.: Overexpression of vimentin: Role in the invasive phenotype in an androgen-independent model of prostate cancer. Cancer Research **63**(9), 2306–2311 (2003)
19. Tseng, P., Yun, S.: A coordinate gradient descent method for nonsmooth separable minimization. Mathematical Programming **117**, 387–423 (2009)
20. Wang, L.P., Wan, C.R.: Comments on "The Extreme Learning Machine". IEEE Transactions on Neural Networks **19**(8), 1494–1495 (Aug 2008)
21. Zhu, C., Byrd, R.H., Lu, P., Nocedal, J.: Algorithm 778: L-BFGS-B: Fortran Subroutines for Large-scale Bound-constrained Optimization. ACM Trans. Math. Softw. **23**(4), 550–560 (Dec 1997)