

Diversity in Extreme Learning Machine ensembles

Carlos Perales-González¹

¹PhD student from Universidad Loyola Andalucía

Thesis directors: Francisco Fernández-Navarro, David Becerra-Alonso, Mariano Carbonero-Ruz

Overview

Introduction

What is machine learning?

Extreme Learning Machine / Ridge classification

Ensembles

State of art

Diverse ensemble

Future work

Machine learning

Machine learning: field of computer science that uses statistical techniques to give computer systems the ability to "learn" (i.e., progressively improve performance on a specific task) with data, without being explicitly programmed [1].

Machine learning is the study of pattern recognition.

- Unsupervised: inferring a hidden structure from unlabeled data. I.e.: Clustering, autoencoders neural networks, ...
- Supervised: learning a function that maps from features to target based on training data. I.e.: SVM, Ridge regression, decision tree, ...
 1. Classification. Target is a category.
 2. Regression. Target is a real value.

Supervised classification

Algorithms minimize the error of classification. How?

- Linear methods.
- Nonlinear methods.
 1. Pure nonlinear methods. I.e.: Decision trees.
 2. Kernels. I.e.: RBF, polynomial.

How is the error function?

- Heuristic. Constrain programming, huge and hard loss functions. I.e.: genetic or greedy algorithms.
- Analytic. Simpler algorithms, based on convex optimization. I.e.: SVM, ELM.

Convex optimization is fast. But **strong assumption**: relation between features and target is convex. **Solution**: ensembles.

ELM I

Extreme Learning Machine for classification, a.k.a. Ridge classification [2], [3], [4].

$$f(\mathbf{x}) = \mathbf{h}'(\mathbf{x}) \boldsymbol{\beta}, \quad (1)$$

where

- $\mathbf{x} \in \mathbb{R}^m$ is the vector of attributes, m is the dimension of the input space.
- $\boldsymbol{\beta} = (\beta_j, j = 1, \dots, J) \in \mathbb{R}^{d \times J}$ is ELM matrix.
- $\mathbf{h} : \mathbb{R}^m \rightarrow \mathbb{R}^d$ is the mapping function and d is the number of hidden nodes (the dimension of the transformed space).

ELM II

Learning problem: Let us also denote \mathbf{H} as

$\mathbf{H} = (h'(\mathbf{x}_i), i = 1, \dots, n) \in \mathbb{R}^{n \times d}$ as the transformation of the training set and $\mathbf{Y} \in \mathbb{R}^{n \times J}$ is the labels "1-of-J" encoded.

$$\min_{\beta \in \mathbb{R}^{d \times J}} \left(\|\beta\|^2 + C \|\mathbf{H}\beta - \mathbf{Y}\|^2 \right), \quad (2)$$

where $C \in \mathbb{R}^+$ is a cross-validated hyper-parameter.

$$\beta = \left(\frac{\mathbf{I}}{C} + \mathbf{H}'\mathbf{H} \right)^{-1} \mathbf{H}'\mathbf{Y} \quad (3)$$

ELM III

It uses:

- Matrix programming.
- Use Tikhonov regularization.

Machine learning problems are not always solvable. To avoid inverse matrix problems, regularization term appears.

Different mapping functions h haven been proposed

- Single Hidden Layer ELM, or Neural ELM.
- Kernel ELM, or Kernel Ridge classification.

Proposed solution to avoid convex assumption: ensembles.

Bagging and Boosting

Bagging (bootstrap aggregating) is a learning method for generating several versions of a base learner by selecting some subsets from the training set and using these as new learning sets [5]. Random, different classifiers.

Boosting is a family of machine learning meta-algorithms which focus on combining base learners over several iterations and generate a weighted majority hypothesis [6].

Our proposal I

Explicit diverse ensembles for classification. This was sent to the International Conference on Hybrid Artificial Intelligent Systems (HAIS) 2018 and accepted.

$$\min_{\beta^l \in \mathbb{R}^{d \times J}} \frac{1}{2} \left(\|\beta^l\|^2 + C \|\mathbf{H}\beta^l - \mathbf{Y}\|^2 + \left(D + \frac{n}{s}\right) \sum_{j=1}^J \sum_{k=1}^{l-1} \langle \beta_j^l, \mathbf{u}_j^k \rangle^2 \right) \quad (4)$$

where:

- $\mathbf{u}^k \in \mathbb{R}^{d \times J}$ is the column-by-column normalized β^k from the iteration k of the ensemble.
- $D > 0$ is a hyper-parameter like C .

Our proposal II

Hence, β_j^l could be obtained analytically as:

$$\beta_j^l = \left(\frac{\mathbf{I}}{C} + \mathbf{H}'\mathbf{H} + \frac{1}{C} \left(D + \frac{n}{s} \right) \mathbf{M}_j^l \right)^{-1} \mathbf{H}'\mathbf{Y}_j \quad j = 1, \dots, J \quad (5)$$

where \mathbf{M}_j^l is defined as

$$\mathbf{M}_j^l \equiv \sum_{k=1}^{l-1} \mathbf{u}_j^k \left(\mathbf{u}_j^k \right)' \quad (6)$$

Results I

	Accuracy (Acc)			
	DELM	AELM	BRELM	NCELM
car	0.929711	0.834618	0.901805	<i>0.905111</i>
winequality-red	0.853687	<i>0.840085</i>	0.839670	0.837363
ERA	0.829479	0.822201	0.828019	<i>0.828428</i>
LEV	0.836345	0.786404	0.792371	<i>0.798220</i>
SWD	0.787940	<i>0.764487</i>	0.759893	0.760442
newthyroid	0.932035	0.817172	0.812035	<i>0.819509</i>
automobile	0.867376	0.834618	0.841636	<i>0.846499</i>
squash-stored	0.694286	0.751429	0.694063	<i>0.711937</i>
squash-unstored	<i>0.814286</i>	0.830952	0.813810	0.812381
pasture	0.833333	0.766667	0.811111	<i>0.826667</i>

Results II

	Diversity (d)			
	DELM	AELM	BRELM	NCELM
car	0.999206	0.180213	0.176621	<i>0.181212</i>
winequality-red	0.926890	0.152451	0.124054	<i>0.185804</i>
ERA	0.968917	0.138991	0.143748	<i>0.156551</i>
LEV	0.992860	0.098013	0.089886	<i>0.133830</i>
SWD	0.980953	0.130116	<i>0.138222</i>	0.137884
newthyroid	0.886023	0.043061	0.040141	<i>0.057340</i>
automobile	0.932272	0.314529	0.311612	<i>0.317588</i>
squash-stored	0.668662	0.216839	0.181023	<i>0.217834</i>
squash-unstored	0.568838	0.130116	0.145780	<i>0.155101</i>
pasture	0.081884	<i>0.181297</i>	0.175300	0.187400

Motivation

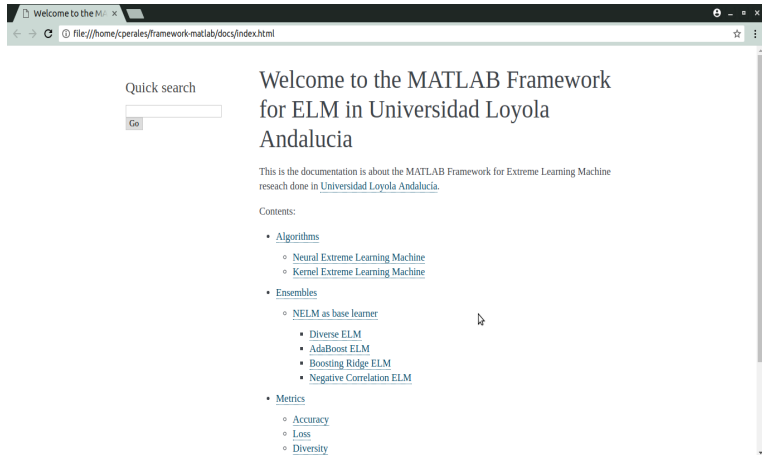
- Solve convex assumption, looking for local convexities.
- Sensitivity analysis. It helps to establish a ranking of good solutions.
- Diverse solutions are looked for in a explicit way.

Work in progress

Improving our proposal to HAIS 2018.

- More ensembles to compare against.
- More data sets.
- Minimizing runtime.
- Sensitivity analysis.

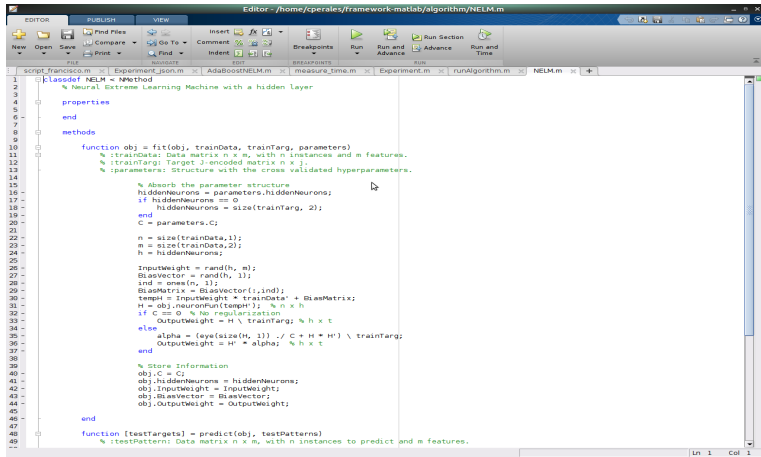
Framework in MATLAB (1)



The screenshot shows a web browser window with the address bar displaying the file path: `file:///home/cperales/framework-matlab/docs/index.html`. The page title is "Welcome to the MATLAB Framework for ELM in Universidad Loyola Andalucía". On the left, there is a "Quick search" section with a text input field and a "Go" button. The main content area features a large heading "Welcome to the MATLAB Framework for ELM in Universidad Loyola Andalucía" followed by a paragraph: "This is the documentation is about the MATLAB Framework for Extreme Learning Machine research done in [Universidad Loyola Andalucía](#)." Below this, a "Contents:" section lists the following items:

- [Algorithms](#)
 - [Neural Extreme Learning Machine](#)
 - [Kernel Extreme Learning Machine](#)
- [Ensembles](#)
 - [NELM as base learner](#)
 - [Diverse ELM](#)
 - [AdaBoost ELM](#)
 - [Boosting Ridge ELM](#)
 - [Negative Correlation ELM](#)
- [Metrics](#)
 - [Accuracy](#)
 - [Loss](#)
 - [Diversity](#)

Framework in MATLAB (2)



```

1 classdef NELM < Method
2     % Neural Extreme Learning Machine with a hidden layer
3
4     properties
5
6     end
7
8     methods
9
10        function obj = fit(obj, trainData, trainTarg, parameters)
11            % :trainData: Data matrix n x m, with n instances and m features.
12            % :trainTarg: Target J-encoded matrix n x J.
13            % :parameters: Structure with the cross validated hyperparameters.
14
15            % Absorb the parameter structure
16            hiddenNeurons = parameters.hiddenNeurons;
17            if hiddenNeurons == 0
18                hiddenNeurons = size(trainTarg, 2);
19            end
20            C = parameters.C;
21
22            n = size(trainData,1);
23            m = size(trainData,2);
24            h = hiddenNeurons;
25
26            InputWeight = rand(h, m);
27            BiasVector = rand(h, 1);
28            ind = ones(n, 1);
29            BiasMatrix = BiasVector(ind);
30            tempH = InputWeight * trainData' + BiasMatrix;
31            H = obj.neuronFun(tempH); % n x h
32            if C == 0 % No regularization
33                OutputWeight = H \ trainTarg; % h x t
34            else
35                alpha = (eye(size(H, 1)) ./ (C + H * H')) \ trainTarg;
36                OutputWeight = H' * alpha; % h x t
37            end
38
39            % Store Information
40            obj.C = C;
41            obj.hiddenNeurons = hiddenNeurons;
42            obj.InputWeight = InputWeight;
43            obj.BiasVector = BiasVector;
44            obj.OutputWeight = OutputWeight;
45
46        end
47
48        function [testTargets] = predict(obj, testPatterns)
49            % :testPattern: Data matrix n x m, with n instances to predict and m features.
50

```


Framework in Python (1)

<https://github.com/cperales/PyRidge>

The screenshot shows a web browser window displaying the PyRidge documentation. The browser's address bar shows the URL <https://cperales.github.io/PyRidge/>. The page has a dark blue sidebar on the left with the PyRidge logo and a search bar. The main content area is white and features the title "Welcome to PyRidge's documentation!" with a "Docs" breadcrumb and an "Edit on GitHub" link. Below the title, there are two status boxes: "Build: passing" and "Coverage: 95%". The text describes the project as a Python-based machine learning library for Ridge classification, mentioning its availability on GitHub and its relation to Extreme Learning Machine. It also discusses the project's motivation, its academic background at Universidad Loyola Andalucía, and a list of features organized into Generic classes, Algorithms, and Functions.

Welcome to PyRidge's documentation! — PyRidge documentation - Google Chrome

Es seguro | <https://cperales.github.io/PyRidge/>

PyRidge

Search docs

Generic classes
Algorithms
Functions

Docs » Welcome to PyRidge's documentation! [Edit on GitHub](#)

Welcome to PyRidge's documentation!

Build: passing Coverage: 95%

This project is aimed to write an useful machine learning library in Python based on the Ridge classification algorithms. Public repository is [available in Github](#). These algorithms can be known in the literature also as Extreme Learning Machine, and they are explained as a type of a feedforward neural network where some neurons does not require to be tuned by calculating them. Algorithms are the same and they achieve to a reasonably good solution.

These supervised machine learning algorithms can be known in the literature as Ridge Classification, Tikhonov regularization or Extreme Learning Machine. A nice discussion about first and second terms can be seen in [this discussion in StackExchange](#).

Although ELM is a polemic topic due to the accusations of plagiarism ([see more here](#) and [here](#)), some actual research is done by applying ensemble techniques to Ridge Classification, thus some some papers are used for implementing algorithms.

Main motivation of this repository is translating from MATLAB to Python 3 what [I am](#) doing in my PhD in Data Science in [Universidad Loyola Andalucía](#).

The library is organized in the following way:

- [Generic classes](#)
 - [Classifier](#)
 - [Neural Method](#)
 - [Kernel Method](#)
- [Algorithms](#)
 - [Neural Ridge Classifier](#)
 - [AdaBoost Neural Ridge Classifier](#)
 - [Kernel Ridge Classifier](#)
- [Functions](#)
 - [Cross validation](#)
 - [Metrics](#)

Framework in Python (2)

```

pyridge - [-/pruebas/pyridge] - ~/pyridge/algorithm/neural.py - PyCharm Community Edition 2017.2.4
File Edit View Navigate Code LaTeX Refactor Run Tools Mathematica VCS Window Help

pyridge pyridge algorithm neural.py
Project: pyridge
Structure:
  pyridge
    build
    config
    data
    data2
    dist
    docs
    env
    profile
    pyridge
      algorithm
        __init__.py
        adaboost.py
        kernel.py
        neural.py
        generic
        utils
          __init__.py
          cross_val.py
          metric.py
          preprocess.py
          save_load.py
          target_encode.py
          __init__.py
        pyridge.egg-info
        saved_clf
        test
          __init__.py
          test_adaboost.py
          test_coverage.py
          test_json.py
          test_kridge.py
          test_neural.py
        work_with_data
          coverage
          coverage.yml
          gitignore
          travis.yml
          LICENSE
  README.md
  TODO
  Python Console
  Terminal
  Version Control
  IDE and Plugin Updates: PyCharm Community Edition is ready to update.

6 class NRidge(NeuralMethod):
7     """
8     Neural Ridge classifier, also known as Extreme Learning Machine.
9     It works as a single hidden layer neural network where
10     neuron's weights are chosen randomly.
11     """
12     __name__ = 'Neural Ridge classifier'
13
14     def fit(self, train_data, train_target):
15         """
16         Use some train (data and target) and parameters to
17         fit the classifier and construct the rules.
18         """
19         iparam numpy.array train data: data with features.
20         iparam numpy.array train target: targets in j codification.
21
22         self.t = train_target.shape[1]
23
24         n = train_data.shape[0]
25         m = train_data.shape[1]
26         h = self.hidden_neurons
27
28         # h x m
29         self.input_weight = np.random.rand(h, m)
30         # h x 1
31         self.bias_vector = np.random.rand(h, 1)
32         bias_matrix = np.resize(self.bias_vector.transpose(),
33                                (n, h)).transpose()
34
35         # h x n
36         temp_H = np.dot(self.input_weight,
37                          train_data.transpose()) + bias_matrix
38         # n x h
39         H = self.neuron_fun(temp_H.transpose())
40
41         if self.C == 0: # Means no regularization
42             # Usually np.linalg.solve gives an error
43             H_inv = np.linalg.pinv(H)
44             self.output_weight = np.dot(H_inv, train_target)
45         else:
46             alpha = np.eye(H.shape[0]) + \
47                     self.C * np.dot(H, H.transpose())
48             self.output_weight = np.dot(alpha,
49                                         np.linalg.solve(alpha,
50                                                         train_target))
51
52     def predict(self, test_data):
53         """
54         """
55
56         NRidge: set_params()

```

scikit-learn / sklearn

The screenshot shows a GitHub pull request titled "Kernel Ridge Classification #10633" by user cperales. The pull request is open and aims to merge 80 commits from the branch "cperales:kernelRidgeClassification" into "scikit-learn:master". The repository has 2,082 watchers, 27,105 stars, and 13,633 forks. The pull request has 129 conversations, 89 commits, and 11 files changed. A comment by cperales, dated 14 Feb, discusses the implementation of Kernel Ridge Regression, explaining its advantages over SVM and providing mathematical details.

Kernel Ridge Classification #10633

Open cperales wants to merge 80 commits into `scikit-learn:master` from `cperales:kernelRidgeClassification`

Conversation 129 Commits 89 Files changed 11

cperales commented on 14 Feb • edited •

Reference Issues/PRs

What does this implement/fix? Explain your changes.

Kernel Ridge Regression (the only Kernel Ridge estimator is right now implemented in master) has a real simple mathematical formulation to adjust a regression problem using squared error loss combined with l2 regularization. It is similar in SVM is done, but with equal constrictions instead of inequal ones as in SVM.

$$\text{Minimize : } L_{P_{\text{RLM}}} = \frac{1}{2} \|\beta\|^2 + C \frac{1}{2} \sum_{i=1}^N \xi_i^2$$

$$\text{Subject to : } h(\mathbf{x}_i)\beta = t_i - \xi_i, \quad i = 1, \dots, N. \quad (23)$$

Using a vector encoding, this regressor can be adapted as a multilabel classifier while keep all the advantages over SVM. And this is my proposal. I followed the mathematical explaining from section 3 in [Venkatesh's paper from 2007](#). Basically, it transforms the target with r labels into a vector with r components.

Any other comments?

This pull request comes after a controversial one about Kernel Extreme Learning Machine (#10602). As @amueller pointed, ELM is so similar to Kernel Ridge that it is possibly a scam because it Huang presents in 2012 ELM as a new algorithm. But Kernel Ridge, as it is presented in `scikit-learn`, just allows regression. This is a modification, using `sklearn.kernel_ridge.KernelRidge` as base, to allow Kernel Ridge multilabel Classification. Name, tests, code and implementation is open to discussion.

1

Reviewers

jnothman

Assignees

No one assigned

Labels

None yet

Projects

None yet

Milestone

No milestone

Notifications




Unsubscribe

You're receiving notifications because you authored the thread.




9 participants

Allow edits from maintainers.

References I

-  A. L. Samuel, "Some studies in machine learning using the game of checkers," *IBM Journal of research and development*, vol. 3, no. 3, pp. 210–229, 1959.
-  A. E. Hoerl and R. W. Kennard, "Ridge regression: Biased estimation for nonorthogonal problems," *Technometrics*, vol. 12, no. 1, pp. 55–67, 1970.
-  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, 2006.

References II

-  G.-B. Huang, H. Zhou, X. Ding, and R. Zhang, "Extreme learning machine for regression and multiclass classification," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 42, no. 2, pp. 513–529, 2012.
-  L. Bbeiman, "Bagging Predictors," *Machine Learning*, vol. 24, pp. 123–140, 1996.
-  Y. Freund and R. E. Schapire, "A Short Introduction to Boosting," *Journal of Japanese Society for Artificial Intelligence*, vol. 14, no. 5, pp. 771–780, 1999.

END

THANK YOU