Cracking the Vinho Verde Wine Data: Extreme Learning Machines vs. Boosted Decision Trees

Kristian Hartikainen (222956) kristian.hartikainen@aalto.fi

Risto Vuorio (84525R) risto.vuorio@aalto.fi

Abstract

In this paper machine learning methods for classification and regression are overviewed. An experimental evaluation of the methods is carried out on the Vinho Verde dataset. The classification methods used are k-nearest-neighbor classifier, bagged decision trees and support vector machines. Multivariate linear regression and extreme learning machines are used for regression. A theoretical overview of the methods is given. Implementations for k-NN classifier, multivariate linear regression and elm are explained. The best performing classification method was the bagged decision trees classifier and the best performing regression method the extreme learning machine.

I. Introduction

Machine learning is a branch of computer science that seeks to use example data and past experience to improve algorithm performance [1]. In this paper machine learning methods are evaluated using the Vinho Verde dataset [2]. More specifically various machine learning methods are used to perform classification and regression tasks.

Machine learning is a broad subject of research and its applications have garnered a lot of interest from the industry in recent years. This paper does not seek to provide in-depth analysis of the best possible methods on the given tasks but rather provide an experiment backed overview to couple of popular methods used in modern machine learning. A good overview to the basic machine learning methods and their mathematical grounding is given by Alpaydin in [1]. The Alpaydin book covers most of the methods used in this paper except for neural networks based extreme learning machines. Neural networks are discussed in [3].

Experiments are conducted in this paper to demonstrate the performance and ease of use of different machine learning algorithms. The algorithms for this paper were selected with two goals in mind. The first goal was to find well performing algorithms for the course competition. The support vector machines and tree bagging were selected purely for their good performance in the dataset. The second goal was to select couple of machine learning algorithms that are both used in real world applications and easy to implement. K-nearest neighbor classifier, linear regression and extreme learning machines were selected for this purpose.

Neural networks have gained a lot of popularity lately as they have performed well in many real world machine learning problems as well as in widely used benchmarks such as the MNIST dataset [3–5]. Extreme learning machine was chosen to represent neural networks in this task as it is easy to implement and is based on similar mathematical model as the best performing neural networks.

The paper is laid out in five sections plus appendices. This introduction is the first section after which the methods used in this paper are explained in section II. Description of the conducted experiments follows in section III. After the experiment decription the results are given in section IV and discussed in section V.

The algorithm implementations are given in the appendices.

II. Methods

A couple of different algorithms were trained for both of the prediction tasks. The algorithms used have been discussed on the T-61.3050 - Machine Learning: Basic Principles course with the exception of extreme learning machines. In this section a theoretical overview of the used learning algorithms and other methods used is given.

I. k-Nearest Neighbor Classifier

The k-Nearest Neighbor (k-NN) classifier is a nonparametric classification technique, which is (generally) used under the assumption that the underlying distribution of the data is unknown. With parametric classification methods, predictions of the unseen data is based on the *fixed parameter models* constructed from the input data. With nonparametric methods, however, the parameter count and nature vary together with the changing data, and the only assumption we can make is the similarity between the input and output data. Various nonparametric methods are available for different machine learning tasks, e.g. classification and regression. [1]

Nonparametric classification algorithms make the decision based the *similarities* between the points to be classified and training data. The definition of the similarity may change between the classification task. One example of similarity measure is the euclidean distance between two points: the closer the points the more similar they are.

The k-NN classifier is a special case of a more general nonparametric classifier. Let us first consider the general nonparametric classification problem. Given a set of N training points $\mathbf{X} = \{x^t\}, t = 1...N$, each belonging to a class ω_i, we want to assign a point $\mathbf{x} \notin \mathbf{X}$ to one of those classes. In k-NN classifier, the input point is assigned to the class with the most points among the neighbors of the input. Figure 1 shows an example of the classification boundaries generated by the k-Nearest Neighbor classifier with a single neighbor (k=1).

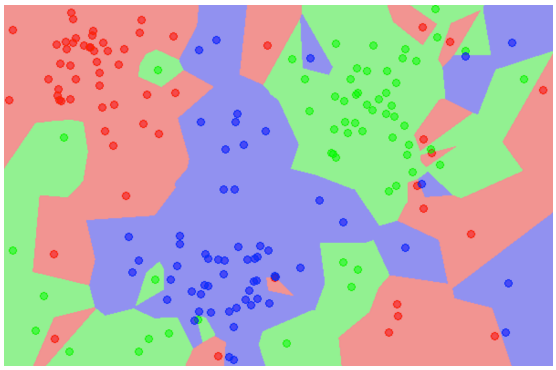

Figure 1: *An example of the classification boundaries of a k-Nearest Neighbor, using a single neighbor (k=1). [6]*

In spite of simplicity, the k-NN classifier is very robust. It is, however, very reliant on the training data, computationally heavy, and unreliable with higher dimensional data. We believe that the k-NN classifier is suitable for the wine classification task. The size of the training dataset seems large enough to produce meaningful results, and at the same time computationally manageable even with naive implementation. Data pre-processing techniques might be needed to handle the dimensionality and other problems related to the similarity measure. These techniques are discussed in subsection VI.

II. Linear Regression

In linear regression the relationship between the input variables $\mathbf{X}$ and the predicted variable $\mathbf{y}$ is modeled using a linear combination of basis functions [1]. The mathematical expression describing the relationship of the input variables and the predicted variable is given in the equation 1 where input variables are denoted by $\mathbf{X}$, the predicted variable is expressed as a function of input variables $g_i(\mathbf{X})$, ϕ are the basis functions and w_j are the weights for each basis function. The closed form solution for the weights is found using linear algebra. The

least squares solution for the weights is given by $\boldsymbol{w} = \boldsymbol{y}\boldsymbol{\Phi}^{\dagger}$ where $\boldsymbol{\Phi}$ is a matrix with elements $\phi(x^t)$ and $\boldsymbol{\Phi}^{\dagger}$ denotes the pseudo inverse of the matrix [1].

$$g_i(\mathbf{X}) = \sum_{j=1}^{k} w_j \phi_{ij}(\mathbf{X}) \tag{1}$$

When doing regression with single input variable polynomial basis functions are often used. For more than one input variable polynomial regression is rarely used, instead linear basis functions are used. [1]

III. Extreme Learning Machine

Artificial neural networks are a family of machine learning models which consist of multiple locally interconnected, simple computing units called neurons, that together form a network type structure. The neurons can be used to form a non-linear mapping from the input space and dividing a complex problem into simpler subproblems. [3]

The output of the network is adjusted by changing the synaptic weights between the neurons. To obtain the desired behavior of the network, the weights are learned by a learning algorithm. Figure 2 shows an example of a single layer feedforward Neural Network. [3]

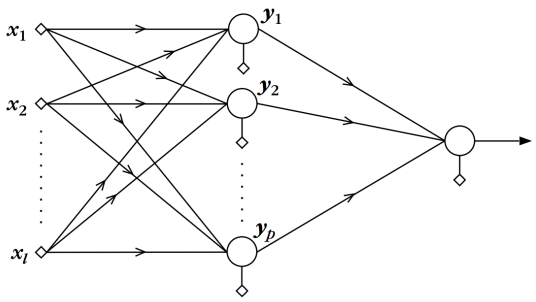

Figure 2: *A single hidden layer Neural Network. [7]*

Extreme Learning Machines are neural network constructs with two special properties. First, they are single layer feedforward networks, meaning that the neurons connection do not form cycles. Secondly, the weights of the hidden layer are randomly assigned and frozen. These properties have interesting consequences. The learning algorithm of the weights reduces to a single step, essentially amounting to finding a solution for a linear system. While this results in learning thousands of times faster than traditional backpropagation algorithm [3], these models can still produce good generalization performance and learn non-trivial, non-linear mappings between the inputs and outputs. [8]

Despite the intriguing results shown in the original paper [8], there has been some controversy in the machine learning community, regarding the extreme learning machines and its author. [9]

IV. Support Vector Machines

Support Vector Machines (SVM) are non-probabilistic supervised learning models that can be used for both classification and regression task. In this paper, we consider only the SVM's used for binary classification.

In the general case, the SVM is used to find the parallel hyperplanes that maximize the marginal between the linearly separable classes, minimizing the classification error. Figure 3 shows an example of a problem solvable by general SVM. [7]

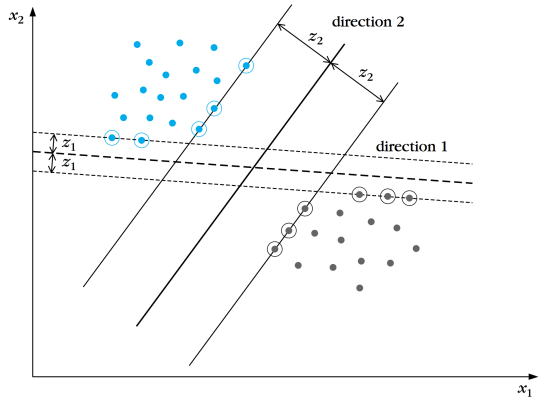

Figure 3: *An example of a linearly separable binary classification problem with two possible linear classifiers. Support Vector Machine finds the hyperplane that maximises the marginal between the classes. [7]*

The general SVM can be extended to solve a non-separable classification problem, by allowing the inputs points to be incorrectly classified with certain cost. Further, a non-linear separation is achieved by non-linearly mapping the input features into a new feature space and using a similar SVM as in the linearly separable case. An example of the non-linear case is presented in the Figure 4. [7]

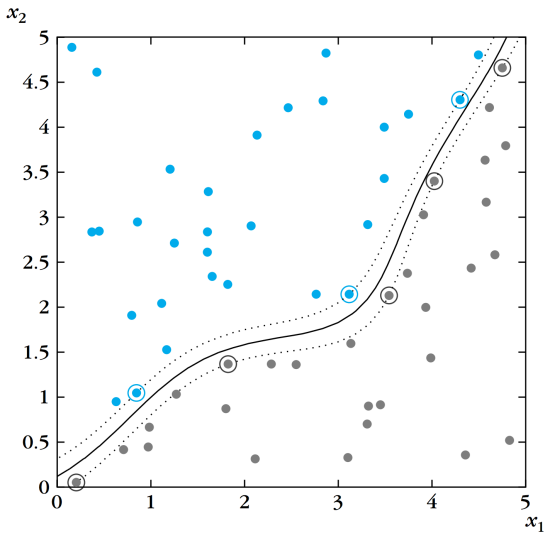

Figure 4: *An example of a classification of non-linear classification done by support vector machine. [7]*

V. Bagged Decision Trees

One method for creating well performing learners is to combine multiple weaker learners in a way that improves the robustness of the model [1]. Combinations of a large number of simpler learners have performed well in machine learning competitions [10]. TreeBagger is a Matlab tool that combines decision trees using bootstrap aggregation [11].

The base learning algorithm used by TreeBagger is decision tree. Decision trees are a supervised nonparametric learning method. Decision trees represent the hypothesis by a tree where each leaf node corresponds to an area in the input space. The tree branches are binary decisions. The learning algorithm uses a heuristic method to select the best feature to branch by. The key idea of learning decision trees is to take advantage of the recursive property of trees that a new tree is created by attaching a tree to a leaf of another tree. [1]

The empirical error approximates the generalization error the better the larger the dataset is. Because the size of the real training data is limited, the training set is enlarged artificially. One way to artificially enlarge the dataset is to use the histogram of the training data as the "true" dataset. Using the histogram for enlarging the dataset is called bootstrap aggregation [12]. TreeBagger uses bagging to create the training datasets for the decision trees.

VI. Data Pre-Processing

A crucial part of successful prediction using machine learning methods is to make sure the data is in suitable form for the learning algorithm used. For example algorithms that depend on a measure of distance between the datapoints, k-NN for example, can generalize the data better when the distances are standardized. The distances are standardized by dividing the features by their standard deviations. [1].

Some learning algorithms benefit from the data being centered around zero. The centering is achieved by subtracting the mean of the data from all of the data vectors. [1]

It is often beneficial to keep the models simple. It is stated that simpler models are more robust on small datasets [1]. Feature selection is one way to reduce model complexity. Feature selection means selecting a subset of the data features for training the predictor. Different methods for feature selection can be used, but they are based on the same idea of repeatedly evaluating validation error on different subsets of features. The subsets can be chosen for example by starting from no features and adding the feature that improves performance the most. One way for feature selection is searching through all the combinations of features but it is often not practical due to large number of features. [1]

VII. Model Evaluation

To make informed decisions about how to optimize the model parameters and in general how to create good predictors, methods for comparison of the created models are needed. A simple measure of the empirical error on the training data is usually not enough. To get more accurate estimates of the generalization error, the training dataset is split into training and validation datasets. The validation dataset is not used for training of the model, as its sole purpose is to be used for estimation of the generalization error. [1]

The number of labeled training samples is limited in real world machine learning problems. Therefore the size reduction to the training dataset resulting from the use of the validation set is not desirable. K-fold cross validation is a commonly used method for validation without drastic change in the training set size. In k-fold cross validation the training dataset is divided to k "folds" one of which is used for validation at a time and the rest for training. Each fold is used in turn for validation the generalization error estimate is the average of the errors for each of the k-folds. [1]

Leave-one-out error is a variation of k-fold error where k is set to one. Leave-one-out error is often computationally prohibitively expensive.

III. Experiments

In the experiments two prediction tasks were studied. The first task was to classify the vinho verde dataset to red and white wines. In the second task wine qualities were approximated with regression. In this section the conducted experiments are described.

I. Wine Type

The wine type prediction was a binary classification task with two classes, namely red and white wines. Two classification methods namel k-nearest-neighbor classifier and support vector machines were studied. A k-nearest-neighbor classifier was implemented from scratch. The Matlab implementation of support vector machines was used. The experiments conducted with the two classifiers are described in the following.

I.1 K-Nearest-Neighbor Classifier

A K-Nearest-Neighbor classifier was implemented. The implementation details are given in appendix VI. In the model selection phase different parameters were experimented with and the results were validated using 20-fold cross validation. The only preprocessing step for the data was centering, meaning making the data zero mean and unit variance.

The parameter space for a model as simple as kNN classifier is large. To limit the required work and computation time, only the most influential parameters were experimented with. The number of neighbors k was the most influential parameter. Values of k from 1 to 50 were tested. Another parameter that was experimented with was the distance function. Euclidean distance and Minkowski distance were tested.

I.2 Support Vector Machine

Support Vector Machines were studied for the classification task. Matlab has a versatile SVM implementation included in the default distribution [13]. The Matlab SVM implementation has a multitude of parameters to tune and the total parameter space is huge. To help reduce the number of parameters to be tested, the Matlab classification toolbox was used to find coarse parameters.

Three different kernel functions were used. Quadratic and cubic kernel functions performed poorly compared to the gaussian kernel functions and they were excluded from further experiments. The gaussian kernel functions were studied using by setting the box constraint to 1 and testing the kernel scale γ with values in the range 0.1-20.

The data was standardized by centering it to zero mean and unit variance. 20-fold cross validation was used for the model selection.

II. Wine Quality

Wine quality prediction was defined as a regression task where a quality rank from 1 to 7 was to be predicted for the white and the red wines. Multivariate linear regression and extreme learning machine algorithms were implemented for the regression task. From the kaggle competition it was learned that better results for the quality prediction task were achieved by treating it as a classification problem. Bootstrap aggregation of decision trees was tested as the classification method.

II.1 Multivariate Linear Regression

Multivariate regression was used to predict the quality labels for the wines. The linear regression model was kept as simple as possible using only linear kernel functions and no data preprocessing. The results were rounded to the nearest integer. With the simplicity as the main goal, there was only one model to test which had 12 parameters that were acquired from the analytical solution for linear regression explained in II. The implementation of the multivariate linear regression model is given in the appendix VII.

II.2 Bootstrap Aggregation with Decision Trees

The Matlab TreeBagger tool was used to create bagged decision trees. The quality prediction was interpreted as a classification task for the experiments using TreeBagger. Regression trees were also tried but they performed poorly compared to the classification models and were excluded from further experiments.

The TreeBagger supports multiple modes of fitting the decision trees and has many parameters for controlling the ensembling of the trees. A cursory overview of the parameter space was conducted by using the Matlab classification toolbox and some manual trials. In the preliminary search for parameters it was found that the default settings of TreeBagger yield the best results and thus further experiments with the various parameters except the number of grown trees and feature selection were omitted.

The feature selection was conducted by trying out all combinations of three or more features. With the final feature selection the number of trees was varied from 1 to 5000. The TreeBagger built-in out-of-bag error (OOBError) was used for model selection. The OOBError computes the misclassification probability for the classification trees in the training data.

II.3 Extreme Learning Machine

Regression with extreme learning machines (ELM) was used for predicting the wine qualities. The ELM was implemented for the experiment by the research group. The ELM implementation is presented in VIII.

The ELM implementation has 11 linear neurons for each of the input variables. The hyperbolic tangent sigmoid transfer function, called tansig in matlab [14], was used as the nonlinear activation function. ELMs were trained using 0 to 1000 nonlinear neurons. The model selection was conducted using leave-one-out validation.

The data was standardized to zero mean and unit variance for the experiments.

IV. Results

The results for the wine type and wine quality predictions are presented in the Tables 1 and 2, respectively.

Table 1: *Results for the wine type predictions*

Method	Parameters	Accuracy [%]
k-NN	Euclidean, k=6	94.0
SVM	Gaussian, $\gamma = 2.85$	99.125

Table 2: *Results for the wine quality predictions*

Method	Parameters	Accuracy [%]
Linear Regression	-	49.75
ELM	l=11, nl=1	54.75
Bagged Trees	250 trees	68.25

For the wine type prediction task, the k-Nearest Neighbor classifier achieved 94.0% accuracy, using euclidean distance and k=6 nearest neighbors. The best prediction accuracy for Support Vector Machine, 99.125%, was achieved using Guassian kernel function with the kernel parameter $\gamma = 2.85$.

For the wine quality prediction task, the Bagged Decision Trees outperformed the Extreme Learning MAchine and the linear regression model. The final extreme learning machine consisted of 11 linear neurons, and 1 non-linear neurons, achieving 54.75% accuracy. The linear regression, with no specific hyperparameters, achieved 49.75% accuracy, and the Bagged Decision Trees, using 250 trees, achieved 68.25% accuracy.

V. Discussion

An interesting finding from the quality prediction task was that the distinction between classification and regression is not always clear. A quality score from 1 to 7 was to be predicted on each of the wines. Intuitively predicting the wine qualities seems like a regression task. However classification type learners performed consistently better than regression learners in the task. It seems there is no easily determined continuous relationship in the data features corresponding to the wine quality.

Next the findings on each of the studied algorithms are briefly discussed.

I. Extreme Learning Machine

The ELM performance was somewhat weaker than what was expected based on the success of neural network methods in the real world machine learning problems. There are many possible causes for the bad performance. One possibility is that in the ELM experiments the model overfitted the data badly. The overfitting could have been combated with weight regularization which was not implemented for the ELM in this paper.

Implementing ELM is easy compared to other neural network methods since no back-propagation or other learning method is needed for the weights of the nonlinear layer. Therefore ELM was a good find for getting a taste of neural networks even though its performance was suboptimal.

II. K-Nearest Neighbor Classifier

K-nearest neighbor classifier performed worse than even very simple support vector machines and thus received little attention for parameter optimization. K-NN is interesting as it is a simple nonparametric method. The amount of training data was possibly too small for the k-NN classifier to perform well.

III. Support Vector Machines

Support vector machines performed well in the type prediction task. The final results could have been better if more careful cross-validation scheme was implemented. This was apparent in the competition submission of the team where the SVM type prediction performed very well (1st place) in the public leaderboard but then got much worse results for the rest of the test set.

IV. Multivariate Linear Regression

Multivariate linear regression was too simple for the given regression task. Considering the structure of the data this should not be surprising.

V. Bootstrap Aggregated Decision Trees

TreeBagger yielded the best results in the quality prediction task. Bagging seems to be a powerful method for these types of prediction tasks as it is often used in machine learning competitions and performed well here.

There were some accuracy problems with the TreeBagger as well. The OOBError that was used for evaluation of the models only measures the error in the training set without using validation. Implementing a validation scheme would have benefited the competition entry.

VI. Conclusion

Conducting both regression and classification on the Vinho Verde dataset with diverse selection of algorithms was a good way to get introduced in the practice of machine learning. The research group started building intuition to the performance and complexity of the algorithms as well as selecting the right tools for the job.

References

[1] E. Alpaydin, *Introduction to machine learning*. MIT press, 2004.

[2] P. Cortez, A. Cerdeira, F. Almeida, T. Matos, and J. Reis, "Modeling wine preferences by data mining from physicochemical properties," *Decision Support Systems*, vol. 47, no. 4, pp. 547–553, 2009.

[3] S. Haykin, *Neural Networks and Learning Machines*, ser. Neural networks and learning machines. Prentice Hall, 2009, no. nid. 10.

[4] L. Wan, M. Zeiler, S. Zhang, Y. L. Cun, and R. Fergus, "Regularization of neural networks using dropconnect," in *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, 2013, pp. 1058–1066.

[5] D. Ciresan, U. Meier, and J. Schmidhuber, "Multi-column deep neural networks for image classification," in *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*. IEEE, 2012, pp. 3642–3649.

[6] Wikipedia. (2015) k-nearest neighbors algorithm. [Online]. Available: https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm

[7] S. Theodoridis and K. Koutroumbas, *Pattern recognition*. Burlington, MA ;: Academic Press, 2009., previous ed.: San Diego, CA : Academic Press, c2006.

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

[9] Reddit. (2015) Yann lecun: What's so great about "extreme learning machines"? [Online]. Available: https://www.reddit.com/r/MachineLearning/comments/34u0go/yann_lecun_whats_so_great_about_extreme_learning/

[10] Kaggle. (2015) Profiling top kagglers: Gilberto titericz, new 1 in the world. [Online]. Available: http://blog.kaggle.com/2015/11/09/profiling-top-kagglers-gilberto-titericz-new-1-in-the-world/

[11] MathWorks. (2015) Treebagger. [Online]. Available: http://se.mathworks.com/help/stats/treebagger.html

[12] L. Breiman, "Bagging predictors," *Machine learning*, vol. 24, no. 2, pp. 123–140, 1996.

[13] MathWorks. (2015) fitcsvm. [Online]. Available: http://se.mathworks.com/help/stats/fitcsvm.html

[14] ——. (2015) tansig. [Online]. Available: http://se.mathworks.com/help/nnet/ref/tansig.html

VI. Appendix A: Source Code for k-Nearest Neighbor Classifier

This section presents the source code of the k-Nearest Neighbor classifier used for the wine type prediction.

```
function [ totalAcc, predictions ] = myKnn(trainX, trainY, testX, k, folds)
%MYKNN Simple k-nearest neighbor classifier
%   Cross-validate the classification accuracy using the training data,
%   i.e. trainX and trainY. Predict the classes for testX.

trainN = size(trainX, 1);
X = standardize(trainX);
Y = zeros(trainN, 1);
classes = unique(trainY);

for i=1:length(classes)
    Y(strcmp(trainY, classes(i))) = i;
end

% Takes as an input N1 x D matrix X1 and N2 x D matrix X2.
% returns a N2 x K matrix, including k nearest neighbours in X
% for each point in Y.
% Find the nearest neighbors f

idx = crossvalind('Kfold',trainN,folds);

totalAcc = 0;
for fold=1:folds
    validIdx = (idx == fold);
    trainIdx = ~validIdx;

    X1 = X(trainIdx, :);
    Y1 = Y(trainIdx, :);
    X2 = X(validIdx, :);
    Y2 = Y(validIdx, :);

    % find the neighbors for the validation points
    distances = pdist2(X1, X2);
    [d, I] = sort(distances);
    neighbors = I(1:k, :)';

    % for each validation points, find the most frequent classes of the
    % neighbors
    predicted = mode(Y1(neighbors), 2);
    acc = sum(predicted == Y2) / length(predicted);
    totalAcc = totalAcc + acc;
end
totalAcc = totalAcc / folds;
```

```
% find the neighbors for the validation points
distances = pdist2(trainX, testX);
[d, I] = sort(distances);
neighbors = I(1:k, :)';

testKey = mode(Y(neighbors), 2);
predictions = classes(testKey);

end
```

VII. Appendix B: Source Code for Linear Regression

This section presents the source code of the linear regression used for the wine quality prediction.

```
function [ testPreds ] = myRegression(trainX, trainY, testX)
%MYREGRESSION Simple linear regression

[Ntrain, dataDim] = size(trainX);

% we are substituting the different degrees with different dimensions of
% linear regressors
basisFunsTrain = ones(Ntrain, dataDim+1);
basisFunsTrain(:, 2:end) = trainX;

betasTmp = basisFunsTrain \ trainY;

% test data
Ntest = size(testX, 1);

basisFunsTest = ones(Ntest, dataDim+1);
basisFunsTest(:, 2:end) = testX;

testPreds = round(basisFunsTest * betasTmp);

end
```

VIII. Appendix C: Source Code for Extreme Learning Machine

This section presents the source code of the Extreme Learning Machine used for the wine quality prediction.

```
function [ totalAcc, predictions ] = myElm( trainX, trainY, testX, ...
                                            lM, nlM )
%MYELM Simple extreme learning machine
%
%    lM - the number of linear neurons
%    nlM - the number of non-linear neurons
```

```
SIGMA = @tansig; % non-linear activation function

X = standardize(trainX)';
Y = trainY';
testX = standardize(testX)';

[n0, Ntrain] = size(X); % number of samples and dimensions

% create random weights
lW    = eye(n0+1);
lW(lM:end, :) = 0; % set the bias linear weight to 0
nlW = unifrnd(-3, 3, [nlM, n0+1]); % for uniform -3...3

biasX = [X; ones(1, Ntrain)];
H = [lW * biasX;
     SIGMA(nlW * biasX);
     ones(1, Ntrain)]';

pseudoinv = pinv(H);
beta = pseudoinv * Y';
P = H*pseudoinv;

e_loo = (Y' - P*Y') ./ (ones(length(Y'), 1) - diag(P));
e = (e_loo' * e_loo) / Ntrain;

Ntest = size(testX, 2);
biasTestX = [testX; ones(1, Ntest)];
Htest = [lW * biasTestX;
     SIGMA(nlW * biasTestX);
     ones(1, Ntest)]';

predictions = beta' * Htest';
totalAcc = 1-e;
end
```