Statistical Machine Learning and Dissolved Gas Analysis: A Review (Appendix)

Piotr Mirowski, *Member, IEEE,* and Yann LeCun

This is the online appendix to our paper submitted to *IEEE Transactions on Power Delivery*, entitled "Statistical Machine Learning and Dissolved Gas Analysis: A Review". It presents the 15 algorithms used in our study, with more technical details than in the paper. Matlab code for all the machine learning libraries, as well as the public Duval dataset, are available at http://cs.nyu.edu/~mirowski/pub/dga.

I. Methods for Classifying DGA Measurements

This section focuses on our statistical machine learning methodology for transformer failure prediction. We begin by formulating the problem from two possible viewpoints: classification or regression (Section I-A). Then we recapitulate the most important concepts of predictive learning in Section I-B before giving high-level overviews of selected classification and regression algorithms[1], in Sections I-C and I-D respectively, along with rationale for their application to DGA. We also evoke two semi-supervised algorithms that can exploit unlabeled DGA data points in Section I-E.

A. A Classification or Regression Problem

1) Formulation as a Binary Classification Problem: Although DGA can diagnose multiple reasons for transformer failures [1]–[3] (e.g. high-energy arcing, hot spots above $400^o\mathtt{C}$, or corona discharges), the primordial task can be expressed as binary classification: "is the transformer at risk of failure?" From a dataset of DGA measures collected on the pool of transformers, one can identify DGA readings recorded shortly before failures, and separate them from historical DGA readings from transformers that kept on operating for an extended period of time. We use the convention that measurement i is labeled $y_i = 0$ in the "faulty" case and $y_i = 1$ in the "normal" case. In the experiments described in the paper, we arbitrarily labeled DGA measurement $\mathbf{x_i}$ as "normal" if it was taken at least 5 years prior to a failure, and "faulty" otherwise.

2) Classifying Measurements Instead of Transformers: As a transformer ages, its risk of failure should increase and the DGA measurements are expected to evolve. Our predictive task therefore shifts from "transformer classification" to "DGA measurement classification", and we associate to each measurement $\mathbf{x_i}$ taken at time t, a label y_i that characterizes the short-term or middle-term risk of failure relative to time t. In the experiments described in the paper, some transformers had more than a single DGA measurement taken across their lifetime (e.g. $\mathbf{x_i}, \mathbf{x_{i+1}}, \ldots$), but we considered the datapoints $(\mathbf{x_i}, y_i), (\mathbf{x_{i+1}}, y_{i+1}), \ldots$ separately.

3) Formulation as a Regression Problem: The second dataset investigated in the paper also contained the time-stamps of DGA measurements, along with information about the time of failure. We used this information to obtain more informative labels $y_i \in [0, 1]$, where $y_i = 0$ would mean "bound to fail", $y_i = 1$ would mean "should not fail in the foreseeable future", and values y_i between those two extremes would quantify the risk of failure. A predictor trained on such dataset could have a real-valued output that would help prioritize the intervention by the utility company[2].

4) Labeled Data for the Regression Problem: The exact method that we used to obtain the labels for the regression task is the following. First, we gathered for each DGA measurement, both the date at which the DGA measurement was taken, and the date at which the corresponding transformer failed, and computed the difference in time, expressed in years. Transformers that had their DGA samples done at the time of or after the failure were given a value of zero, while transformers that did not fail were associated an arbitrary high value. These values corresponded to the Time-To-Failure (TTF) in years. Then, we considered only the DGA samples from transformers that (ultimately) failed, and sorted the TTF in order to compute their empirical Cumulated Distribution Function (CDF). TTFs of zero would correspond to a CDF of zero, while very long TTFs would asymptotically converge to a CDF of one. The CDF can be simply implemented using a sorting algorithm; on a finite set of TTF values, the CDF value itself corresponds to the rank of the sorted value, divided by the number of elements. Our proposed approach to obtain labels for the regression task of the Time-to-failure (TTF) is to employ the values of the CDF as the labels. Under that scheme, all "normal" DGA samples from transformers that did not fail (yet) are simply labeled 1.

B. Generalities

Before delving into more detailed descriptions of learning algorithms, let us explicit their commonalities.

1) Supervised Learning of the Predictor: Supervised learning consists in fitting a *predictive* model to a training dataset $(\mathbf{X}, \mathbf{y})$ (which consists here in pairs $\{(\mathbf{x_i}, y_i)\}$ of DGA measurements $\mathbf{x_i}$ and associated risk-of-failure labels y_i). The objective is merely to optimize a function f such that for each data point $\mathbf{x_i}$, the prediction $\bar{y}_i = f(\mathbf{x_i})$ is as close as possible to the ground truth *target* y_i. In classification tasks, the discrepancy E between all $\bar{y}_i$ and y_i is generally quantified as a sum of errors $\sum_i 1_{\bar{y}_i \neq y_i}$ or as joint probability $P(\bar{\mathbf{y}} = \mathbf{y}|f) = \prod_i P(\bar{y}_i = y_i|f)$. In regression tasks, E is most often defined as the total square error $\sum_i \| \bar{y}_i - y_i \|_2^2$.

2) Training, Validation and Test Sets: Good statistical machine learning algorithms are capable of extrapolating knowledge and of generalizing it on unseen data points. For this

P. Mirowski is with the Statistics and Learning Research Department of Alcatel-Lucent Bell Laboratories, Murray Hill, NJ, 07974, USA. Web page: http://www.mirowski.info.

Y. LeCun is with the Courant Institute of Mathematical Sciences, New York University.

[1]Matlab code for all the machine learning libraries is available at http://cs.nyu.edu/~mirowski/pub/dga

[2]Note that many classification algorithms, although trained on binary classes, can provide with probabilities.

reason, we separate the known data points into a *training* (*in-sample*) set, used to define model f, and a *test* (*out-of-sample*) set, used exclusively to quantify the predictive power of f.

3) Selection of Hyper-parameters by Cross-validation: Most models, including the non-parametric ones, need the specification of a few *hyperparameters* (e.g. the number of nearest neighbors, or the number of hidden units in a neural network); to this effect, a subset of the training data (called the *validation* set) can be set apart during learning, in order to evaluate the quality of fit of the model for various values of the hyperparameters. In our research, we resorted to *cross-validation*, i.e. multiple (here 5-fold) validation on five non-overlapping sets. More specifically, for each choice of hyper-parameters, we performed five cross-validations on five sets that contained each 20% of the available training data, while the remaining 80% would be used to fit the model.

C. Classification techniques

Before dwelling into descriptions of 15 classification and regression algorithms employed in this study, and for ease of explanation, we illustrate on Fig. 1 how a few classification and regression techniques behave on two-dimensional data. We trained six different classifiers or regressors on a two-dimensional, two-gas training set D_{tr} of real DGA data (that we extracted from the seven-gas Duval public dataset, and we plot on Fig. 1 failure prediction results of each algorithm on the entire two-gas DGA subspace (in grayscale values ranging from white for "normal" transformers, $\bar{y} = 1$, to black for "faulty" transformers, $\bar{y} = 0$). Some algorithms have a linear decision boundary at $\bar{y} = 0.5$, while other ones are non-linear, some smoother than others. For each of the six algorithms, we also report the accuracy on the *training* set D_{tr}. Not all algorithms fit the training data D_{tr} perfectly; as can be seen on these plots, some algorithms obtain very high accuracy on the training set (e.g. 100% for k-Nearest Neighbors), whereas their behavior on the entire two-gas DGA space is incorrect; for instance, very low concentrations of both DGA gases, here standardized $\log_{10}(\mathtt{CH_4})$ and $\log_{10}(\mathtt{C_2H_4})$ with values below -1.5, are classified as "faulty" (in black) by k-NN. The explanation is very simple: real DGA data are very noisy and two DGA gases (namely $\mathtt{CH_4}$ and $\mathtt{C_2H_4}$ in this example) are not enough to discriminate well between "faulty" and "normal" transformers. For this reason, we see on Fig. 1 "faulty" datapoints (red crosses) that have very low concentrations of $\mathtt{CH_4}$ and $\mathtt{C_2H_4}$, lower than "normal" datapoints (blue circles): those faulty datapoints may have other gases at much higher concentrations, and we most likely need to consider all seven DGA gases (and perhaps additional information about the transformer) to discriminate well. This figure should also serve as a cautionary tale about the risk of a statistical learning algorithm that overfits the training data but that generalizes poorly on additional test data.

1) k-Nearest Neighbors (k-NN): k-NN [4] is perhaps the simplest non-parametric classification technique. For a given data sample $\mathbf{x}$, one defines the Euclidian distance $d_i = ||\mathbf{x}-\mathbf{x}_i||_2$ between $\mathbf{x}$ and each data point $\mathbf{x}_i$ in the training set. Based on that metric, k-NN selects the k nearest neighbors of $\mathbf{x}$; the predicted label $\bar{y}$ is the one which is dominant among the k neighbors. k is a hyper-parameter that needs to be optimized on a validation set.

The intuition for using k-NN classifiers in the case of DGA data can be described as "reasoning by analogy": to assess the risk of a given DGA measurement we compare it to the most similar DGA samples in the database. Fig. 1 (top left) shows an example of the k-NN predictions using only two input variables (here gases).

2) C-45 Decision Trees: Decision trees implement a sequence of binary decision rules on each feature [5]. For instance, the k-th node in the binary decision tree might perform the following test on the j-th feature of the i-th sample: "$\mathtt{IF}(x_{i,j} > \theta_k)\mathtt{THEN}\ldots\mathtt{ELSE}\ldots$", and branch out to the "yes" or to the "no" sub-tree, according to the answer. In the case of DGA, decision trees can be likened to the tables of limit concentrations used in [3] to quantify whether a transformer has dissolved gas-in-oil concentrations below safe limits. Instead of pre-determined key gas concentrations or concentration ratios, all these rules are however automatically *learned* from the supplied training data: i.e. that the number of nodes in the tree, along with its layout the choice of the features $x_{i,j}$ and the thresholds θ_k in the previous example are all adjustable parameters. C-45 did not need further hyper-parameters[3].

3) Logistic Regression: Linear classifiers can be seen as a special case of a decision tree with a single node, where the decision boundary is not an individual feature, but a linear combination of the M input features, each of them weighted by associated *regression coefficients* β_j (see Eq. 1). In the M-dimensional space, each sample $\mathbf{x}$ is projected onto the axis co-linear to vector $(\beta_1, \beta_2, \ldots, \beta_M)^T$, with offset β_0. That axis is orthogonal to the hyperplane defined by Eq. 1. The binary decision is made simply by taking the sign (-1 or +1) or the Kronecker δ (0 or 1) of the projection (Eq. 2).

$$f(\mathbf{x}) = \sum_{j=1}^{M} \beta_j x_j + \beta_0 \quad (1)$$

$$\bar{y} = \delta_{f(\mathbf{x})>0} \quad (2)$$

Note that in this work, where DGA data are log-normalized, the linear classifier defines a planar boundary in the log-space of DGA data, but that this boundary is no longer planar in the natural DGA space.

The decision taken by the linear classifier can be expressed as a conditional probability distribution on the output label y, given the data sample $\mathbf{x}$ (Eq. 3). Learning a logistic regression classifier consists then in maximizing the probability of the predicted label being correct for all the training data points. We notice that this probability of the label being 1 goes from 0 (when $f(\mathbf{x})$ is very negative), to 0.5 (when $f(\mathbf{x}) = 0$) and finally to 1 (when $f(\mathbf{x})$ tends to infinity).

$$p(y = 1|\mathbf{x}) = \frac{1}{1 + e^{-f(\mathbf{x})}} \quad (3)$$

4) Regularized Logistic Regression: Among parametric models, regularization [6] is a heuristic aiming at ensuring that the trained model does not *overfit* the training data and that it *generalizes* well to unseen test data. This heuristic assumes that, all other things being equal, a model where the parameters have a smaller overall magnitude might overfit less. Regularization therefore corresponds to a penalty on the norm of the parameter vector (e.g. on vector $(\beta_0, \ldots, \beta_M)$ in the case of linear or logistic regression). In a probabilistic framework, regularization corresponds to imposing a prior on small values of the parameters (regression coefficients). A hyperparameter λ controls how much importance is given to the regularization constraint over the quality of fit to the data. Two popular norms are used for regularization: the Euclidian

[3]Matlab code taken from http://www.yom-tov.info/Uploads/C4_5.m.

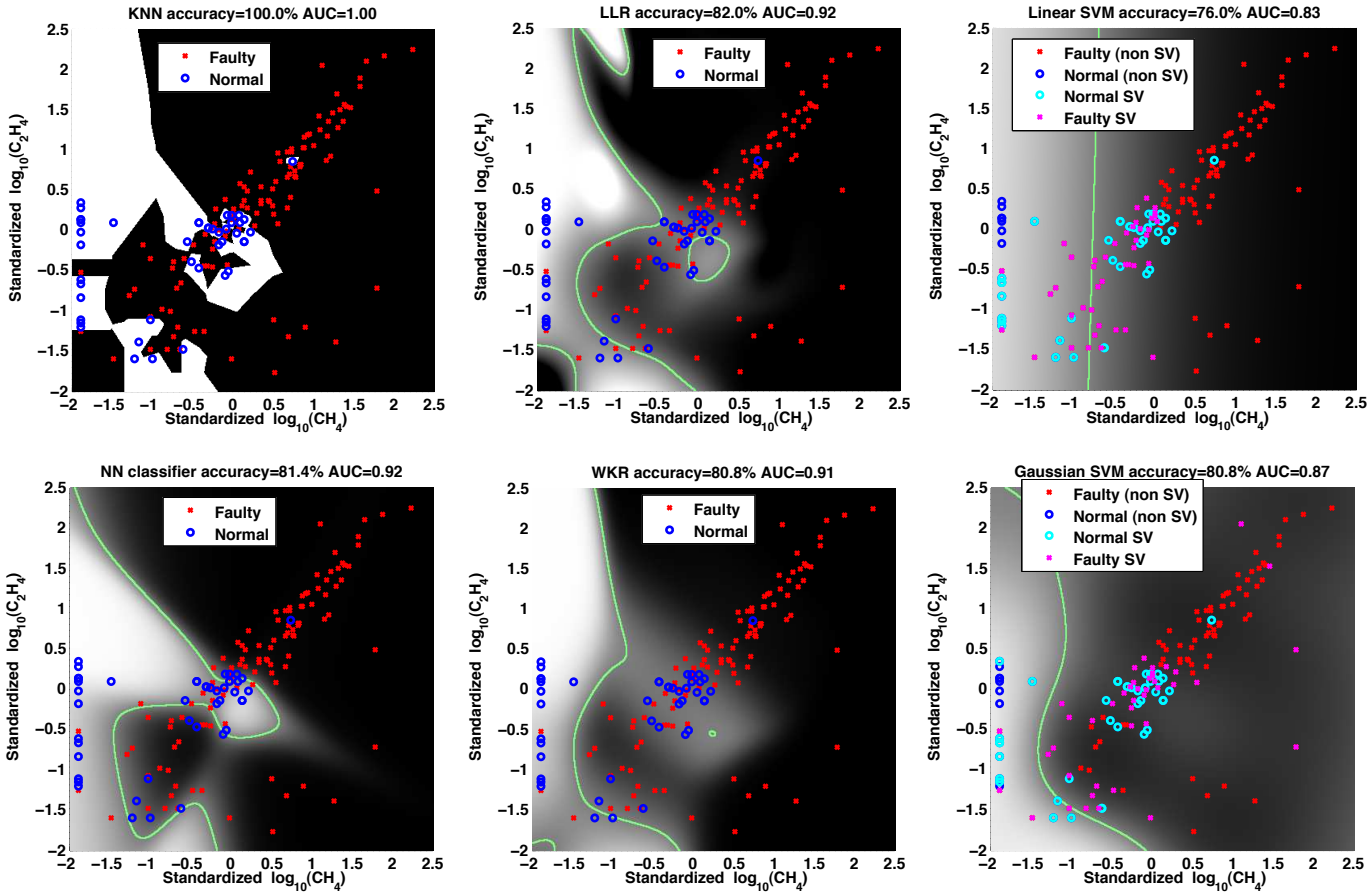

Fig. 1. Comparison of 6 regression or classification techniques on a simplified two-dimensional version of the Duval dataset (see paper) consisting of log-transformed and standardized values of dissolved gas analysis measures for CH_4 and C_2H_4. There are 167 datapoints: 117 "faulty" DGA measures (marked as red or magenta crosses) and 50 "normal" ones (blue or cyan circles). Because the training datapoints are not easily separable in 2D, the accuracy and Area Under the Curve (see paper) on the training set are generally not 100%. The test data points consist in the entire DGA values space. Output of the 6 decision functions goes from white ($\bar{y} = 1$, meaning "no impending failure predicted") to black ($\bar{y} = 0$, meaning "failure is deemed imminent"); for most classification algorithms, we plot the continuously-valued probability of having $\bar{y} = 1$ instead of the actual binary decision ($\bar{y} = 0$ or $\bar{y} = 1$). The decision boundary (at $\bar{y} = 0.5$) is marked in green. Note that we do not know the actual labels for the test data - this figure provides instead with an intuition of how the classification and regression algorithms operate. K-Nearest Neighbors (KNN, top left) partitions the space in a binary way, according to the Euclidian distances to the training datapoints. Weighted Kernel Regression (WKR, bottom middle) is a smoothed version of KNN, and Local Linear Regression (LLR, top middle) performs linear regression on small neighborhoods, with an overall nonlinear behavior. Neural Networks (bottom left) cut the space into multiple regions. Support Vector Machines (SVM, right) use only a subset of the datapoints (so-called support vectors, in cyan and magenta) to define the decision boundary. Linear kernel SVMs (top right) behave like logistic regression and perform linear classification, while Gaussian kernel SVMs (bottom right) behave like WKR.

L_2 norm, whose result is that after optimization, the magnitude of the parameters is "spread" across all the variables, and the L_1 norm, that tends to bring some parameters to zero, and thus favors only a small subset of non-zero parameters [7].

In the case of DGA, L_1 norm-regularized logistic regression tends to select only a few key gases in the decision function, setting the value of the parameters for the other gases to zero (thereby ignoring those gases in the classification). This technique is useful to select the most relevant key gases for the transformer failure prediction task.

Linear classifiers in general, including (regularized) logistic regression, all behave in a similar way to the top right plot on Fig. 1 (where we plot $p(y = 1|\mathbf{x})$ for all $\mathbf{x}$ in the 2D plane).

5) Neural Networks: We consider here the Multi-Layer Perceptron (MLP) architecture [8], which corresponds to a neural network with one or more hidden layers (in this study, we restricted ourselves to the single hidden layer architecture). Intuitively, each layer of the neural net contains as many "logistic regressions" as there are hidden nodes/units on that layer, which means that each layer cuts its input space with as many hyperplanes as there are hidden nodes. Each hidden node is followed by a nonlinear function; in our implementation, we used the hyperbolic tangent $\tanh$ that would squish the output between -1 and 1. Other nonlinearities, such as the logistic sigmoid (as in Eq. 3) are possible. Those nonlinearities accentuate the separation made by the decision hyperplane. Each successive layer combines linear combinations of the decision function at each hidden node, and as an end-result, the neural network defines its decision function using a partition of the input feature space into multiple subspaces or polyhedrons. Fig. 1 (bottom left) shows how a two-layer neural network with 10 hidden nodes defines a highly nonlinear decision boundary.

Learning the neural network classifier, i.e. adjusting the parameters ("weights") of the neural network, is done by error back-propagation [8], [9]. The particular embodiment of our training algorithm was stochastic gradient descent [10] with a small momentum weight of 0.01 and with a learning rate $\eta = 0.1$ annealed by a factor of 0.99 every pass on the dataset, trained until the validation error starts increasing. We cross-validated two hyperparameters: the regularization weight λ and the number of hidden units.

6) Support Vector Machines (SVM): Support Vector Machines [11] are a recent, popular and efficient statistical learning tool that can be qualified as mostly non-parametric. They rely on the definition of a kernel function $k(\mathbf{x}, \mathbf{x}_i)$ that can be seen as a measure of symmetric "similarity" between the two samples $\mathbf{x}$ and $\mathbf{x}_i$. The decision function $\bar{y} = \delta_{f(\mathbf{x})>0}$ for a sample $\mathbf{x}$ is defined in terms of the kernel function between $\mathbf{x}$ and the data points in the training set (as in Eq. 4); it involves a minimal, sparse, set of support vectors $\{\mathbf{x}_i\}_{i\in S}$ that are each given a weight α_i. Learning in SVM corresponds to finding a minimal set S of support vectors that minimizes the error on the training labels.

$$f(\mathbf{x}) = \sum_{i\in S} \alpha_i y_i k(\mathbf{x}, \mathbf{x}_i) \tag{4}$$

We considered 3 different kernels in this study: the linear kernel, the polynomial kernel and the Gaussian kernel; see Fig. 1 (right) for a comparison between linear and Gaussian kernels. The linear kernel ($k(\mathbf{x}, \mathbf{x}_i) = \mathbf{x}^T\mathbf{x}_i$) enables us to express the learning of a linear classifier as an SVM problem; in our experiments, we use this algorithm instead of logistic regression. The polynomial kernel (we used second-order polynomials) adds a (parabolic) curvature to the decision surface. Finally, the Gaussian kernel defines a distance-based similarity metric between data samples:

$$k(\mathbf{x}, \mathbf{x}_i) = e^{-\gamma d_i^2} = e^{-\gamma||\mathbf{x}-\mathbf{x}_i||_2^2} \tag{5}$$

SVMs are also called *maximum margin* classifiers, because their decision boundary is, by construction, as far as possible from the training data points, so that they remain well separated according to their labels. Maximum margin training enables better generalization of the classifier to unseen examples, and it is equivalent to L_2-norm regularization [12]. We cross-validated the SVM's regularization coefficient C as well as the Gaussian spread coefficient γ.

D. Regression of the Time-to-Failure

1) Linear Regression and LASSO: Linear regression is based on the same principles as linear classification, but instead of classifying the sign of the prediction $f(\mathbf{x})$ (see Eq. 1), one uses the prediction as is: $\bar{y} = f(\mathbf{x})$. Fig. 1 (top right) essentially shows an example of linear regression. Similarly to the classification case, an L_2-norm or L_1-norm regularization can be imposed on the regression coefficients $(\beta_0, \ldots, \beta_M)$ (respectively corresponding to *ridge regression* [6] or to the LASSO [7]).

In terms of DGA, a linear regression model establishes a linear dependency between every single variable/regressor (here log-concentrations) x_j and the target/predicted labels y (respectively $\bar{y}$).

2) Weighted Kernel Regression (WKR): Weighted kernel regression [13] is the simplest among the non-parametric regression algorithms, and it is the continuously-valued equivalent of the k-NN algorithm (see Fig. 1, bottom). Once a distance d_i is computed between the sample $\mathbf{x}$ and each data point $\mathbf{x}_i$ in the training set, it is used in a Gaussian kernel function (Eq. 5) (where the "spread" coefficient γ is a hyper-parameter) to compute the weight of data point $\mathbf{x}_i$. The decision function is a weighted interpolation over all the training dataset:

$$\bar{y} = \frac{\sum_{i=1}^{N} k(\mathbf{x}, \mathbf{x}_i) y_i}{\sum_{i=1}^{N} k(\mathbf{x}, \mathbf{x}_i)} \tag{6}$$

WKR assumes a smoothness within the input data, and in our case, claims that two transformers that have similar DGA measures should face similar risks of failure.

3) Local Linear Regression (LLR): Local linear regression interpolates a prediction $\bar{y}$ to a sample $\mathbf{x}$ by fitting a parametric linear regression model to the *neighborhood* of $\mathbf{x}$ in the Euclidian space [14]. It therefore both benefits from the locality of WKR and from the regressor dependency of linear regression. See Fig. 1 (top) for an example of LLR in 2D.

4) Neural Network Regression: Just like a linear regressor is a linear (logistic) classifier without the binary decision, a neural network regressor can be seen as a neural network where the outputs of the last, predictive, layer are continuously-valued and not categorical. We otherwise used the same architecture and the same training algorithm as for the classifier. The output of the decision function of a neural net regressor can actually be used as input to Eq. 3, i.e. to the probability of having $y = 1$ in a neural net classifier.

5) Support Vector Regression (SVR): Although support vector regression presents a few subtleties compared to the SVM classification algorithm [15] (in particular, one specifies the tolerable margin of error hyper-parameter ϵ, which we set to 0.01), the main principles (kernel method, maximum margin classifier, optimization) remain the same.

E. Semi-Supervised Algorithms

In presence of large amounts of unlabeled data (as was the case for the utility company's dataset explained in the paper), it can be helpful to include them along the labeled data when training the predictor. The intuition behind Semi-Supervised Learning (SSL) is indeed that the learner could get better preparation for the test set "exam" if it knew the distribution of the test data points (aka "questions"). Note that the test set labels (aka "answers") would still not be supplied at training time.

1) Semi-Supervised Classification: Low Dimensional Scaling (LDS) [16] builds on kernel methods and SVMs for binary classification. When no label information is available, it exploits the distance (in the feature space) between data points to build a graph of nearest neighbors. It then uses the graph to find low density regions (i.e. regions of the feature space where fewer data points are present) in order to place the binary decision boundary in those low density regions. We chose this algorithm because it obtained state-of-the-art results on various real-world datasets (see the comparative study [17]), and we used an out-of-the box implementation of LDS[4], setting the Gaussian kernel spread to $\sigma = 1$, and cross-validating the regularization penalty C and the LDS-specific graph exponent ρ.

2) Semi-Supervised Regression: Finally, we also considered an SSL algorithm for regression and chose Local Linear Semi-supervised Regression (LLSSR) [18]. LLSSR extends Local Linear Regression, with an additional assumption that the value of the regression function should not change suddenly. In other words, we hypothesize that DGA samples that are close to each other should map to similar risks of failure.

REFERENCES

[1] M. Duval, "Dissolved gas analysis: It can save your transformer," *IEEE Electrical Insulation Magazine*, vol. 5, pp. 22–27, 1989.

[2] M. Duval and A. dePablo, "Interpretation of gas-in-oil analysis using new IEC publication 60599 and IEC TC 10 databases," *IEEE Electrical Insulation Magazine*, vol. 17, pp. 31–41, 2001.

[4]Matlab implementation of LDS available at http://olivier.chapelle.cc/lds/.

TABLE I
SUMMARY OF THE STATISTICAL LEARNING METHODS EMPLOYED.

Algorithm	Objective	Linear	Local	Parametric
k-NN	classif.	no	yes	no
C4.5	classif.	no	no	yes
SVM linear	classif.	yes	no	yes
SVM quad.	classif.	no	no	yes
SVM gauss.	classif.	no	yes	yes
NN	classif.	no	no	yes
LDS (semi-supervised)	classif.	no	yes	yes
Linear regression	regress.	yes	no	yes
LASSO regression	regress.	yes	no	yes
NN	regress.	no	no	yes
SVR quad.	regress.	no	no	yes
SVR gauss.	regress.	no	yes	yes
WKR	regress.	no	yes	no
LLR	regress.	no	yes	yes
LLSSR (semi-supervised)	regress.	no	yes	yes

[3] *IEEE Guide for the Interpretation of Gases Generated in Oil-Immersed Transformers*, IEEE Std. C57.104-2008, 2009.

[4] T. Cover and P. Hart, "Nearest neighbor pattern classification," *IEEE Transactions on Information Theory*, 1967.

[5] J. Quinlan, *C4.5: Programs for machine learning*. Morgan Kaufman, 1993.

[6] A. Tychonoff and V. Arsenin, *Solution of Ill-posed Problems*. Washington, DC: Winston & Sons, 1977.

[7] R. Tibshirani, "Regression shrinkage and selection via the lasso," *Journal of the Royal Statistics Society*, vol. 58, pp. 267–288, 2006.

[8] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, *Learning internal representations by error propagation*. Cambridge, MA, USA: MIT Press, 1986, pp. 318–362.

[9] Y. LeCun, L. Bottou, G. Orr, and K. Muller, "Efficient backprop," in *Lecture Notes in Computer Science*. Berlin/Heidelberg: Springer, 1998.

[10] L. Bottou, "Stochastic learning," in *Advanced Lectures on Machine Learning*, O. B. et al. (Eds.), Ed. Berlin Heidelberg: Springer-Verlag, 2004, pp. 146–168.

[11] C. Cortes and V. Vapnik, "Support vector networks," *Machine Learning*, 1995.

[12] V. Vapnik, *The Nature of Statistical Learning Theory*. Berlin: Springer-Verlag, 1995.

[13] E. Nadaraya, "On estimating regression," *Theory of Probability and Its Applications*, vol. 9, pp. 141–142, 1964.

[14] C. Stone, "Consistent nonparametric regression," *Annals of Statistics*, vol. 5, pp. 595–620, 1977.

[15] A. J. Smola and B. Schlkopf, "A tutorial on support vector regression," *Statistics and Computing*, vol. 14, pp. 199–222, 2004.

[16] O. Chapelle and A. Zien, "Semi-supervised classification by low density separation," in *Proceedings of the Conference on Artificial Intelligence and Statistics AISTATS'05*, 2005.

[17] A. N. Erkan and Y. Altun, "Semi-supervised learning via generalized maximum entropy," in *Proceedings of the Conference on Artificial Intelligence and Statistics AISTATS'10*, 2010.

[18] M. R. Rwebangira and J. Lafferty, "Local linear semi-supervised regression," School of Computer Science Carnegie Mellon University, Pittsburgh, PA 15213, Tech. Rep. CMU-CS-09-106, Feb. 2009.