Classification of URLs Using Interpretable Machine Learning

William Briguglio[1]

Abstract— **Today, the fast-changing nature of malware websites provides a significant challenge to data security specialists. The increased ability for authors to bi-pass traditional detection methods has forced security specialist to look to machine learning and artificial intelligence for creating robust detection systems. One area of research has been the use of machine learning algorithms for automatically detecting malicious websites. However, due to the nature of these detection systems, false positives and false negatives are not permitted, meaning detection systems must be finely tunable. Thus, it is necessary to know why a decision was made, and this means methods for interpreting machine learning decisions must be employed. In this paper we provide a brief overview of machine learning interpretability methods. We then test various classification algorithms on a data set containing network and application layer features for benign and malicious URLs. Next, we provide explanations for a subset of the tested algorithms. Finally, we finish off with a discussion of our results and necessary future work.**

I. INTRODUCTION

Malware authors design and implement a large variety of malware attacks which use malicious websites as an attack vector. These sites are typically disguised as benign or legitimate in order to trick unsuspecting users into visiting them and triggering some malicious event such as a malware download or the execution of malicious JavaScript code. In many cases these sites will be linked to from a legitimate website that has been compromised, however there are various methods used by malware authors to trick users into clicking a malicious URL. Thus, it is not easy for users to distinguish between malicious or benign URLs, and once the site is visited, it may be too late, and the malicious events already triggered. For this reason, it is important to determine whether a URL links to a benign or malicious site without the user having to visit the site pointed to by the URL.

One method of distinguishing between malicious and benign URLs is to visit the site in a controlled environment (e.g. VM) and click various buttons and links on the site, walking the directory structure, or waiting till a certain amount of time has passed in case a malicious event is triggered after a set amount of time has elapsed. Another is to look at the HTML and JavaScript files served when accessing a site and examining them for any characteristics that may be common to malicious websites, without actually executing any code. Both of these methods have their strengths however, each requires a high degree of interaction with the website, are difficult to implement, and require a large amount of processing overhead.

[1]W. Briguglio is with the School of and Computer Science, University of Windsor, ON N9B3P4, Canada. `briguglw@uwindsor.ca`

A more computationally conservative method with a low degree of client server interactivity would be using a lightweight classification algorithm that could determine if a site is malicious using just application layer and network characteristics. However, methods using classification algorithms are prone to false positives and false negatives, so they require careful fine tuning and adjustments. For this reason, it is also necessary for such a classifier to have some degree of interpretability. Furthermore, in order for explanations generated for the classification decisions to be easily understood, the feature set should be small. A small feature set would also help in making the classifier computationally inexpensive to use.

In this paper we propose an interpretable and computationally lightweight classifier using just network and application layer characteristics. The data set used is the Malicious and Benign Websites data set from Kaggle. The data set contains 1781 labeled samples with 20 different application layer and network characteristics such as URL length, or server operating system. The feature set is first evaluated using various methods to determine the most meaningful and important features. Next the subset of most meaningful features is used to train various classification algorithms using supervised learning. The most effective classifier and feature subset is selected for interpretation. Finally, a discussion of the results and a conclusion is presented.

II. LITERATURE REVIEW

In the last decade machine learning algorithms have seen an explosion in popularity both in the industry and in academia. The cause of this spike in interest can be chiefly attributed to the increase in computational power in the last decade or so. However, due to the massive feature sets these algorithms are now being used on, the prediction process of these algorithms has become so non-intuitive that analysis in the traditional sense (combining knowledge of the models algorithms with visualizations and simple statistics like accuracy or AUC) has become practically impossible. Analysis being necessary for a number of practical and legal concerns has caused research to now shift towards machine learning interpretability.

Christoph Molnar [1] put together a summary of machine learning interpretation methods in which he outlines a basic approach for the interpretation of Linear Regression models (of course the same approach can be applied to linear SVMs, Shirataki et al. [2]) where a features contribution to a prediction is the product of its value and weight. For logistic regression he shows that when the j^{th} feature value is incremented by a value of 1, then the predicted log odds of the sample belonging to the positive class is increased by β_j, where β_j is the weight of feature j. Alternatively, he shows that a unit increase in feature j results in the predicted odds increasing by $(e^{\beta_j} - 1)\%$. He goes on to discuss the seemingly trivial interpretation of decision trees as the conjunction of the conditions described in the nodes along a predictions path to a leaf/prediction node using the AND logical operator. Similarly, for rule list models, an "explanation" is simply re-stating the rule or combination of rules which lead to a decision.

The evaluation of a models complexity is closely tied with its explanations comprehensibility, especially for rule set models, linear models, and tree models. Given the following complexity definitions, the explanation approaches discussed above could be too complex for highly dimensional datasets. Marco Ribeiro et al. [3] define the complexity of a linear model as the number of non-zero weights and the complexity of a decision tree as the depth of the tree. Meanwhile, Otero and Freitas [4] defined the complexity of a list of rules as the average number of conditions evaluated to classify a set of test data. They referred to this as the "prediction-explanation size".

There has also been work done on the interpretability of neural networks(NNs) such as the Lawyer-Wise Relevance Propagation introduced in [5] as a set of constraints. The constraints ensure that the total relevance is preserved from one layer to another as well as that the relevance of each node is equal to the sum of relevance contributions from its input nodes which in turn is equal to the sum of relevance contributions to its output nodes. In [6], Shrikumar et al. propose DeepLIFT which attributes to each node a contribution to the difference in prediction from a reference prediction by back propagating the difference in predication scaled by the difference in intermediate and initial inputs.

Moving on to model agnostic methods, Friedman in [7] used Partial Dependence Plots (PDP) to show the marginal effect a feature has in a predictive model. Similarly, Goldstein et al. [8] used Individual Conditional Expectation (ICE) plots to show a curve for each sample in the data set where one or two features are free variables while the rest of the features remain fixed. Since ICE plots and PDPs do not work well with strongly correlated features, Deniel W. Apley et al. [9] proposed Accumulated Local Effects plots to display the average local effect a feature has on predictions.

The H-statistic was used by Friedman and Popescu in [10] (equations 44-46) to provide a statistical estimate of the interaction strength between features by measuring the fraction of variance not captured by the effects of single variables. Feature Importance was measured by Breiman [11] as the increase in model error after a features values are permuted (a.k.a. permutation importance).

Christoph Molnar in [1] also discusses the use of a surrogate model which is an explainable model that closely approximates a black box model. Marco Ribeiro et al. in [3] defined a version of the previous method which can explain individual predictions using an approach called Local Interpretable Model-agnostic Explanations (LIME) which trains an interpretable classifier by heavily weighing samples nearer to a sample of interest. Tomi Peltola [12] extended this work with KL-LIME, which generated local interpretable probabilistic models for Bayesian predictive models (although the method can also be applied to non-Bayesian probabilistic models) by minimizing the Kullback-Leibler divergence of the predictive model and the interpretable model. This has the added benefit of providing explanations that account for model uncertainty. Strumbelj et al. [13] detailed how to describe the contributions made by each feature to a prediction for a specific instance using Shapely Values, a concept adopted from coalitional game theory.

Finally there are Example-Based methods such as the method put forward by Wachter et al. in [14] which finds counter-factual examples by finding a sample with a significant difference in prediction, whose features are relatively similar to the sample of interest, by minimizing a loss function. The found sample is then used to explain what small changes would cause the original prediction to change meaningfully. There is also the MMD-critic algorithm by Kim et al. [15] which finds Prototypes (well represented examples) and Criticisms (poorly represented examples) in the dataset. To find examples in the training data which have a strong effect on a trained linear regression model (i.e. influential instances) Cook [16] proposed Cook's distance, a measure of the difference in predictions made by a linear regression model (however the measure can be generalized to any model) trained with and without an instance of interest. Koh and Liang [17] put forward a method for estimating the influence of a specific instance without retraining the model as long as the model has a twice differentiable loss function. Tomsett et al. [18] suggested that interpretability is conceptually linked to adversarial examples. They go on to provide a brief discussion of how adversarial examples have been used to improve interpretability of NNs in other works.

There has also been a variety of implementations of the above methods published. For instance, as discussed in [19], H20.ai (an open source big-data analysis software which can interface with python and other programming languages that are popular among data scientists), includes implementations of partial dependence and individual conditional expectation plots, LIME, monotonicity for Gradient Boosting models using the XGBoost library [20], global surrogates using decision trees, and more. The inventors of LIME have also written their own python library [21] which implements their algorithm. There is also ELI5 [22], which includes a text classifier explainer leveraging the LIME algorithm, an implementation of permutation importance, and other explanatory algorithms. There is a the SHAP (SHapley Additive exPlanations) library [23] which implements the Shapely value local interpretation method discussed above as well as some other methods which combine shapely values with LIME and DeepLift. The specifics of the libraries interpretations methods are described in [24] which is by the library authors. Finally there is Skater [25], a python library which has implementations of layer-wise relevance propagation, tree surrogates, feature importance, PDPs, and

LIME, among others.

III. METHOD

Training and classification were done on a data set of feature samples extracted from 1781 URL's. The data set is discussed in the data set author's paper [26] [1]. The dataset contains 1563 benign samples and 216 malicious samples. The features contain various network and application layer characteristics such as URL length or server OS. Before preforming feature selection and classification, the data set was first cleaned. Content length was set to 0 for samples with a content length value of "NaN". Some samples had "NaN" as their value for other categorical features but were removed since there was only two such cases. The date of registration and date of last update features were changed to the number of days since registration and the number of days since last update, respectively. The feature corresponding to state/province was removed since the number of samples per state/province was too small to be an effective feature. Also, since the categorical features were one-hot-encoded, including the state/province feature would cause the number of features to become too large. The rest of the categorical features were included using one-hot-encoding. This caused the number of features to go from 19 to 310. Finally, since many of the servers were the same OS but with different version numbers, engineered features for the four most common server OSs (nginx, Apache, Microsoft, and mw) were added, and one feature (misc) was added, which is set to 1 if the server is none of the four most common. If a sample was any version of one of the four common server OSs, then the corresponding engineered feature would be set to 1. The feature set is discussed in appendix i.

Five classification methods were tested; Logistic Regressions, SVM (polynomial kernel), Naive Bayes, Random Forest, and Neural Networks. For the first four methods, feature selection was performed to select the 20 best features before training the classifiers. To select features for the Logistic regression and Naive Bayes models, F-test was used. This was done using scikits `f_classif()` function. F-test was chosen because it can detect only linear dependencies between a feature and a samples classification. These are the only dependencies which can be accurately modeled with a logistic regression model. Further, Naive Bayes has the assumption that features are independent and thus can't model feature interactions. Therefore, a feature selection method suited to Naive Bayes should not take into account feature interactions. For SVM with polynomial kernel, Mutual Information (MI) was used. This was done using scikits `mutual_info_classif()` function. MI was chosen because it can detect the more complex dependencies between a feature and a sample's classification that a SVM with polynomial kernel or Naive Bayes classifier can model. For the Random Forest model, the recursive feature elimination (RFE) implementation in scikit was used. RFE starts by training a classifier (which uses weighted coefficients or calculates feature importances) on the set of all features. In this case, Random Forest was used since the features found would then be used in a Random Forest Classifier. Then the feature with the least importance is removed from consideration and the process is repeated until all features but one (the most important) is removed. The order of removal of the features then determines their ranking in priority for selection. In the case of the Neural Network, the model was trained on all features, then Layer-wise Relevance Propagation (LRP) was used to calculate the absolute-value mean relevance of the input neurons. Absolute value was used because a highly relevant input neuron can have a large negative or a large positive value, and averageing without taking the absolute value first would cause these values to cancel out and the average to be close to zero. This would make it appear as if the neuron is irrelevant when this is not the case. After the the average neuron relevances were calculated, the twenty best features corresponding to the most relevant neurons were chosen for the final NN model.

[1]The data set was downloaded from `www.kaggle.com/xwolf12/malicious-and-benign-websites`

Finally, to be extra thorough, the models were trained once with each feature set obtained from each of the feature selection methods. This was also done as a cursory investigation of LRP as a method for finding a meaningful (in this case the most relevant) subset of features. Gridsearch with 2-fold cross validation was used to find the best parameters for Logistic Regression and SVM. Given that the primary focus was not classification results but rather the interpretation, an exhaustive search for the other models best paramets was not done. Rather, for Random forest, the model was trained using 100 tress and no max depth as this produced more then satisfactory results. For training the NN, a trial and error approach was used to determine suitable architecture and hyper parameters. The model feature-set pairs were then evaluated and a subset of the models were selected for interpretation. Weighted accuracy was used to account for the class imbalance by weighting the accuracies inversely proportional to the class size. A discussion of results and interpretation follows in the next section

IV. CLASSIFICATION RESULTS

Table I. shows the weighted accuracies of the five models trained on each of the feature subsets generated from the feature selection techniques.

TABLE I
MODEL PERFORMANCES

ML Model	Feature Selection Technique			
	F-score	Mutual Info.	RFE	LRP
SVM(polynomial)	84.8%	87.6%	76.5%	84.3%
Naive Bayes	80.2%	77.5%	78.5%	75.9%
Random Forest	81.1%	82.4%	84.5%	81.7%
Logistic Regression	81.4%	78.7%	76.4%	77.2%
Neural Net	79.0%	78.6%	73.4%	77.1%

The NN trained on all features had an 89.1% weighted accuracy

As expected, the best feature selection method was F-score for Naive Bayes and Logistic Regression, MI for SVM, and

RFE for Random Forest. Interestingly, all feature selection methods preformed close to the same, but never the best, for the Neural Network. This could be due to the complexity of NNs allowing the model to learn any of the relationships present in the feature subsets but simultaneously causing overfitting that doesn't occur when using simpler models that make assumptions about the data set (such as the no feature interaction assumptions made by Naive Bayes and Logistic Regression). Another interesting finding is the effectiveness of the LRP feature set. These results show that using the absolute-value mean relevance of input neurons is an effective way to find meaningful features for tabular based data sets.

V. INTERPRETATION

The two models selected for interpretation were Logistic Regression with F-score since its highly comprehensible and high fidelity explanations can possibly serve as a reference point, and Neural Network with F-score to provide an example of a black box interpretation.

A. Logistic Regression

The interpretation of logistic regression is relatively simple and of high fidelity, meaning it clearly and accurately explains why a prediction was made. Table II shows the weights learnt by the classifier for each feature.

TABLE II
MODEL COEFFICIENTS

Feature Name	β_j	$(1 - e^{\beta_j})\%$
nginx	0.95	1.59
Apache/2.4.23 (Unix) OpenSSL	8.13	3393.8
Apache	-0.27	-0.24
RU	7.07	1175.15
nginx	-0.43	-0.35
None	0.37	0.45
REMOTE_IPS	0.08	0.08
CZ	5.59	266.74
misc	-0.88	-0.59
Microsoft-IIS/6.0	5.82	335.97
Apache/2.2.14 (FreeBSD) mod_ssl	9.99	21806.3
TCP_CONVERSATION_EXCHANGE	2.13	7.41
APP_BYTES	20.9	1193313823
SOURCE_APP_PACKETS	-1.59	-0.8
APP_PACKETS	-1.59	-0.8
US	-0.62	-0.46
REMOTE_APP_BYTES	-24.02	-1
REMOTE_APP_PACKETS	0.75	1.12
DNS_QUERY_TIMES	5.81	332.62
ES	11.77	129313.15
Bias	3.16	22.57

Recall that For logistic regression, when the j^{th} feature value is incremented by a value of 1, then the predicted odds increase by $(e^{\beta_j} - 1)\%$. With this in mind we can see some standout entries. The highest weighted feature is APP_BYTES. We may be tempted to say that for every extra byte received from the server, the odds of being classified as malicious$(y = 1)$ increases by 1193313823%. Of course this does not make since as this would mean every sample would be classified as malicious. However we have scaled the input so that all features have mean 0 and variance of 1. Therefore, the following would be a more correct interpretation. For every increase in the number of bytes, b, received from the server, if $b = \sigma$, where σ is the training standard deviation of the APP_BYTES feature, then the odds of being classified as malicious increase by 1193313823% (see appendix ii). Another way of putting it is, if the number of bytes received from the server is one standard deviation above the training mean, then there is a an approximate 12 million fold increase in the odds of being classified malicious. The model gives a 0.042 odds of malicious classification for a feature vector of all 0's, which corresponds to a sample with the training mean for each feature. Given our interpretation, we can say that all things remaining the same, if the bytes received from the server increase by the training standard deviation, then the new odds of a malicious classification is $0.042 \times 1193313823 = 50119180$. This is not intuitive and the only real insight we can draw from this interpretation is that the model learnt that a large number of bytes received from the server guarantees a malicious URL, which is somewhat consistent with industry knowledge and may serve to build trust in the model with industry professionals but is otherwise not very useful.

We have some categorical features that are much easier to interpret given that these features are not scaled. Our second highest weighted feature is ES, which is 1 if the server location is in Spain, and 0 otherwise. The interpretation of this feature is as follows. If the server location changes to Spain, and all other things remain the same, then the odds of a malicious classification increases by 129313.15%. This does not seem correct since there is no reason to assume that every Spanish URL is malicious, however after reviewing our data set we find that despite only 12% of the samples being malicious, about 70% of samples where ES=1 are malicious. This, in combination with the fact that the malicious samples are weight much higher then the benign samples to account for the class imbalances, caused the model to learn that Spanish URLs are malicious.

B. Neural Network

The interpretation of a Neural Network is much more difficult due to the complexity of the model, with its large amount of parameters and hyper parameters which obfuscates the decision process. One method for interpreting NNs is Layer-wise Relevance Propagation, in which the contribution or "relevance" of the input nodes is determined by back propagating the activation of the output nodes. Typically used to explain single predictions which use images, we use it here to determine the "relevance" of input nodes, and thus the relevance of input features, when classifying a single sample. We also calculate the absolute-value mean relevance across each class in the test set to get an idea of what features are most relevant when classifying a malicious vs a benign example. Table III shows the average relevance's for each class and Table IV shows the relevance's for a single malicious example.

We can see, by comparing the benign and malicious

TABLE III
AVG. FEATURE RELEVANCES

Feature Name	Benign Avg.	Malicious Avg.
nginx	-0.0006496	0.002555989
Apache/2.4.23 (Unix)...	-0.0004739	0.015471565
Apache	0.00154213	-0.035468244
RU	-0.0005455	0
nginx	0.00148554	0.007062163
None	0.00926683	-0.016149375
REMOTE_IPS	-0.0001627	0.016747724
CZ	-0.0307944	0.687277238
misc	0.00238879	0
Microsoft-IIS/6.0	-0.0537775	0.132700513
Apache/2.2.14 (FreeBSD)	0	0.768498556
TCP_CONVER...	0.00048803	-0.117660041
APP_BYTES	-0.0008611	-0.139424988
SOURCE_APP_PACKETS	-0.0012357	-0.001394697
APP_PACKETS	-0.000331	0.003797381
US	0.02696391	-0.44043603
REMOTE_APP_BYTES	-0.0014231	0.184379233
REMOTE_APP_PACKETS	-0.0096519	0.421032559
DNS_QUERY_TIMES	-0.0148894	0.465058154
ES	-0.0004357	0.238723478

TABLE IV
FEATURE RELEVANCES

Feature Name	Malicious Ex.
nginx	0
Apache/2.4.23	0
Apache	0.05096776
RU	0
nginx	0
None	0
REMOTE_IPS	0.05470881
CZ	0
misc	0
Microsoft-IIS/6.0	0
Apache/2.2.14	0
TCP_CONVERSATION_EXCHANGE	0.13644174
APP_BYTES	0.13110884
SOURCE_APP_PACKETS	0.00775442
APP_PACKETS	0.03064873
US	0
REMOTE_APP_BYTES	0.00600234
REMOTE_APP_PACKETS	-0.0401238
DNS_QUERY_TIMES	-0.044956
ES	0.58900487

columns, that the features CZ, Apache/2.2.14 (FREEBSD), REMOTE_APP_PACKETS, DNS_QUERY_TIMES and ES play a important role in determining a sample is malicious. The average value for these features for the malicious samples in the test set are 0.02, 0.08, 0.719888142, 0.942165188, and 0.52 respectively. All these numbers are much lower for the average feature values in the benign samples of the test set. This tells us that although not all samples where CZ and Apache/2.2.14 (FREEBSD) are true are classified as malicious, there are some samples with these features set to true, where they play a very large role in a malicious classification. However, for the latter three features, since the average value is so high (between 0.5 to 1 standard deviation above the mean), we can conclude that the classifier learnt that long DNS query times, large amounts of packets transferred from the server, and server location in Spain are indicators of malicious URLs. These first two learnt facts are supported with industry knowledge and can help build trust in our model. The last two are corroborated with the findings of our first model and although the inference about Spanish servers hosting malicious URLs is false, we can now debug our model so that it learns rules that generalizes better. Looking at our malicious example, we see that the Apache/2.2.14 (FREEBSD) and CZ features had 0 relevance, which is consistent with our findings drawn from the average malicious feature relevances, since in this example these features were set to false. However REMOTE_APP_PACKETS and DNS_QUERY_TIMES actually contribute slightly towards a benign classification despite their values both being high. This could be due to the learnt relationship between these two features not being monotonic, in which case enforcing monotonicity may help improve interpretability at the cost of performance. However, a more likely explanation is that this is a outlier which is malicious but does not have typical malicious feature values and due to overfitting, the model was still able to classify this example correctly. The solution still may be monotonicity as it helps combat the problem of overfitting.

We can also see that the benign feature values' average contribution towards benign classification is somewhat small. The only somewhat notable contribution made by a feature is DNS_QUERY_TIMES which also has an average value of -0.155 for benign examples in the test set, and this is consistent whit our interpretation of the malicious average relevance's. The cause for these weak relevances could be the class imbalance. Since there are so many benign examples, there could be more complex relationships learnt by the model which are non monotonic. This could cause large negative and large positive relevances to be cancelled out during averaging. The class imbalance also means that given a reference example where all inputs are 0, the model will already have a low chance of malicious classification, in our case 12.8%. This means the model doesn't need to learn large weights to ensure a benign classification for benign examples. This is also consistent with the large weights we've seen for some of the logistic regression model features.

Our findings show that normalization, scaling, and not enforcing monotonicity can increase classifier performance but at the cost of easier interpretation. In addition, more accurate models, such as NNs, may have better performance but also lack interpretability relative to simpler models like logistic or linear regression. Furthermore, class imbalances create difficult to interpret results as learnt relationships between the poorly represented class and the feature values are obfuscated by the fact that any sample is already very likely to belong to the well represented class. We've also shown that interpretation methods can help debug overfitting or problems within the data set and provide meaningful general rules for black box models.

VI. CONCLUSION

Our results show that the interpretation of machine learning models in the malicious URL detection domain is a non-trivial task, even for "simple" models like logistic regression. Despite this we've shown that interpretation can be used to find relationships between features and predictions that are consistent with industry knowledge, and can therefore build trust in models. Furthermore we've shown the utility of interpretation in debugging models and finding problems in the training sets. Our results also highlighted the performance/interpretability trade off. Specifically we've shown that more complex models may be more accurate but far less interpretable, and that routine prepossessing techniques like scaling and normalization obfuscate the relationships between predictions and features, making interpretation non-intuitive.

For future work, other algorithms need to be tested thoroughly throughout the computer security domain. ML Interpretation within this domain is a new field and application techniques of current interpretation algorithms, as well the development of new algorithms will be necessary. One specific area of interest could be the applicability of LIME to NNs trained on data sets taken from the computer security domain.

APPENDIX

Appendix i

The 310 features used for feature selection consist of:

- 7 binary features corresponding to the one hot encoding of the original char set feature
- 239 binary features corresponding to the one hot encoding of the original server feature
- 44 binary features corresponding to the one hot encoding of the original country feature
- 5 binary features that were engineered and discussed in section III.
- 2 Numerical features representing the original date features.
- 13 Numerical features which include all 12 numerical features and the content length feature from the original data set.

Appendix ii

The feature weights for scaled features must be interpreted as follows:

"For every prior to scaling increase, i, in the feature value of the j^{th} feature, if $i = \sigma_j$, where σ_j is the training standard deviation of the j^{th} feature, then the odds of being classified as malicious increase by $(e^{\beta_j} - 1)\%$"

The reason being is because, as we have already mention in section II, when the j^{th} feature value is incremented by a value of 1, then the predicted odds increase by $(e^{\beta_j} - 1)\%$, for non-scaled features that is. However for scaled features, the feature is first centered around 0 by subtracting the feature mean from the feature value, then divided by the feature standard deviation. Therefore, a change in the pre-scaled feature value which corresponds to a change of 1 in the scaled feature value would be equal to the standard deviation of the feature.

To illustrate, let a, b and μ be the original pre-scaled value of some feature, the pre-scaled increase in the value of the same feature, and the mean of that feature respectively. Then $(a + b - \mu)/\sigma = (a - \mu)/\sigma + b/\sigma$ where the first term on the right hand side is the original sclaed value of the feature, and the second term the scaled increase in the value of the feature. Here it is easy to see the scaled increase will only be 1 if $b = \sigma$.

REFERENCES

[1] Christoph Molnar, "Interpretable Machine Learning: A Guide for Making Black Box Models Explainable".

[2] Shohei Shirataki, Saneyasu Yamaguchi, "A Study on Interpretability of Decision of Machine Learning". Department of Information and Communications Engineering, Kogakuin University Graduate School, Tokyo, Japan

[3] Marco Tulio Ribeiro, Sameer Singh, Carlos Guestrin, " "Why Should I Trust You?" Explaining the Predictions of Any Classifier". KDD 2016 San Francisco, CA, USA.

[4] Fernando E. B. Otero, Alex A. Freitas, "Improving the Interpretability of Classification Rules Discovered by an Ant Colony Algorithm". GECCO13, July 610, 2013, Amsterdam, The Netherlands.

[5] Sebastian Bach, Alexander Binder, Grgoire Montavon, Frederick Klauschen, Klaus-Robert Muller, Wojciech Samek, "On Pixel-Wise Explanations for Non-Linear Classifier Decisions by Layer-Wise Relevance Propagation". PLoS ONE 10 (7): e0130140. doi:10.1371/journal.pone.0130140, 2015.

[6] Avanti Shrikumar, Peyton Greenside, Anshul Kundaje, "Learning Important Features Through Propagating Activation Differences". Stanford University, Stanford, California, USA.

[7] Jerome H. Friedman, "Greedy Function Approximation: A Gradient Boosting Machine". Sequoia Hall, Stanford University, Stanford, California, USA.

[8] Goldstein, A., Kapelner, A., Bleich, J., and Pitkin, E., "Peeking Inside the Black Box: Visualizing Statistical Learning With Plots of Individual Conditional Expectation". (2014) Journal of Computational and Graphical Statistics, in press

[9] Daniel W. Apley, "Visualizing the Effects of Predictor Variable". Evanston, IL: Northwestern University, Department of Industrial Engineering & Managements Sciences.

[10] Jerome H. Friedman, Bogdan E. Popescu, "Predictive Learning via Rule Ensembles". Department of Statistics and Stanford Linear Accelerator Center, Stanford University, Stanford, CA 94305.

[11] Leo Breiman, "Random Forests. Statistics Department University of California, Berkeley, CA 94720

[12] Tomi Peltola, "Local Interpretable Model-agnostic Explanations of Bayesian Predictive Models via KullbackLeibler Projections". Helsinki Institute for Information Technology HIIT, Department of Computer Science, Aalto University, Helsinki, Finland.

[13] Erik Strumbelj, Igor Kononenko, "An Efficient Explanation of Individual Classifications using Game Theory". Faculty of Computer and Information Science, University of Ljubljana, Trzaska 25, 1000 Ljubljana, Slovenia

[14] Sandra Wachter, Brent Mittelstadt, Chris Russell, "Counterfactual Explanations Without Opening the Black Box: Automated Decisions and the GDPR". Oxford Internet Institute, University of Oxford, Oxford UK and The Alan Turing Institite, Bristish Library, London, UK.

[15] Been Kim, Rajiv Khanna, Oluwasanmi Koyejo, "Examples are not Enough, Learn to Criticize! Criticism for Interpretability". 29th Conference on Neural Information Processing Systems (NIPS 2016), Barcelona, Spain.

[16] R. Dennis Cook, "Detection of Influential Observation in Linear Regression". Technometrics, vol. 19, No. 1, pp. 15-18, February 1977

[17] PangWei Koh, Percy Liang, "Understanding Black-box Predictions via Influence Functions". Proceedings of the 34 th International Conference on Machine Learning, Sydney, Australia, PMLR 70, 2017.

[18] Richard Tomsett, Amy Widdicombe, Tianwei Xing, Supriyo Chakraborty, Simon Julier, Prudhvi Gurran, Raghuveer Rao, Mani Srivastava, "Why the Failure? How Adversarial Examples Can Provide Insights for Interpretable Machine Learning". 2018 21st International Conference on Informtion Fusion (FUSION)

[19] Patrick Hall, Navdeep Gill, Mark Chan, "Practical Techniques for Interpreting Machine Learning Models: Introductory Open Source Examples Using Python, H2O, and XGBoost". H2O.ai, Mountain View, CA, February 3, 2018.

[20] Tianqi Chen, Carlos Guestrin, "XGBoost: A Scalable Tree Boosting System". In 22nd SIGKDD Conference on Knowledge Discovery and Data Mining, 2016. https://github.com/dmlc/xgboost

[21] https://github.com/marcotcr/lime

[22] https://github.com/TeamHG-Memex/eli5

[23] https://github.com/slundberg/shap

[24] Scott M. Lundberg, Su-In Lee, "A Unified Approach to Interpreting Model Predictions". Advances in Neural Information Processing Systems 30 (NIPS 2017).

[25] https://github.com/oracle/Skater

[26] Urcuqui, C., Navarro, A., Osorio, J., & Garca, M. (2017). Machine Learning Classifiers to Detect Malicious Websites. CEUR Workshop Proceedings. Vol 1950, 14-17.