Combination Frequency Differencing

D R Kuhn
M S Raunak
Computer Security Division
Information Technology Laboratory

R N Kacker
Applied and Computational Mathematics Division
Information Technology Laboratory

December 6, 2021

This publication is available free of charge from:
https://doi.org/10.6028/NIST.CSWP.12062021-draft

National Institute of Standards and Technology
U.S. Department of Commerce

Abstract

This paper introduces a new method related to combinatorial testing and measurement, *combination frequency differencing* (CFD), and illustrates the use of CFD in machine learning applications. Combinatorial coverage measures have been defined and applied to a wide range of problems, including fault location and for evaluating the adequacy of test inputs and input space models. More recently, methods applying coverage measures have been used in applications of artificial intelligence and machine learning, for explainability and for analyzing aspects of transfer learning. These methods have been developed using measures that depend on the inclusion or absence of t-tuples of values in inputs, training data, and test cases. In this paper, we extend these combinatorial coverage measures to include the frequency of occurrence of combinations. Combination frequency differencing is particularly suited to AI/ML applications, where training data sets used in learning systems are dependent on the prevalence of various attributes of elements of class and non-class sets. We illustrate the use of this method by applying it to analyzing physically unclonable functions (PUFs) for bit combinations that disproportionately influences PUF response values, and in turn provides indication of the PUF potentially being more vulnerable to model-building attacks. Additionally, it is shown that combination frequency differences provide a simple but effective algorithm for classification problems.

Keywords

combinatorial coverage; combination frequency difference; combinatorial testing; physical unclonable function (PUF); unclonable.

Acknowledgments

The authors are very grateful to Charles Prado of INMETRO Brazil for data used in the PUF analysis and to Sandip Kundu and Vinay Patil for helpful discussion. The authors also plan to continue working with INMETRO to apply and develop these ideas for practical application to PUFs.

Additional Information

For additional information on NIST's Cybersecurity programs, projects and publications, visit the Computer Security Resource Center. Information on other efforts at NIST and in the Information Technology Laboratory (ITL) is also available.

Public Comment Period: *December 6, 2021 through February 7, 2022*

National Institute of Standards and Technology
Attn: Computer Security Division, Information Technology Laboratory
100 Bureau Drive (Mail Stop 8930) Gaithersburg, MD 20899-8930
Email: cfdwp@nist.gov

All comments are subject to release under the Freedom of Information Act (FOIA).

1 Introduction

Methods and tools for measuring combinatorial coverage were initially developed to analyze the degree to which test sets included *t*-way combinations of values (for some specified level of *t*) [1][2][4] and have since been studied extensively in the realm of system and software testing [7][8][9][10][11]. Combinatorial coverage measures have been defined and applied to a wide range of problems, specifically for fault location and for evaluating the adequacy of test inputs and input space models. More recently, coverage measures have been used for explainability in artificial intelligence and machine learning [24][28] and for analyzing aspects of transfer learning [27]. These methods have been developed using measures that depend on the inclusion or absence of *t*-tuples of values in inputs and test cases. For software testing, primarily for deterministic systems where the presence of a particular combination always triggers a specified error, it is relevant whether a *t*-tuple of values is present in test inputs, but the number of occurrences of a particular *t*-tuple of values is generally not relevant to testing. Multiple occurrences are only redundant and do not add value. These measures can also be applied in artificial intelligence and machine learning (AI/ML) systems.

For many aspects of assurance of autonomous systems and machine learning, this type combinatorial coverage measure is valuable and possibly essential, since the correct and safe behavior of many AI systems is dependent on the training inputs. Conventional structural coverage measures are not applicable to such black box behavior. Consequently, it is essential to evaluate the degree to which possible combinations of input attribute values have been included in training and test sets for AI and autonomy. (Attributes in a machine learning setting correspond to parameters in a test effort; they are the inputs to the system.) If the system has not been shown to function correctly for an input combination that may be encountered in use, then assurance is inadequate. However, for some questions in machine learning, consider the frequency (or rate) of occurrence of *t*-tuples of values in input and how two different sets may compare or differ in combinatorial coverage.

This paper applies combinatorial coverage measures from [13], which include the frequency of occurrence of combinations, in an approach referred to as *combination frequency differencing* (CFD). This method is particularly suited to AI/ML applications, where training data sets used in learning systems are dependent on the prevalence of various attributes of elements of class and non-class sets. This paper illustrates the use of this method by applying it to analyzing physical unclonable functions (PUFs) for potential weaknesses in design and showing how it can be extended to develop a simple but effective classification algorithm.

2 Combinatorial Coverage and Combination Frequency Differences

This section reviews the basic measures of combinatorial coverage and applications of these measures in Section 2.1. This idea is extended to measures that include the frequency of occurrence of combinations in Section 2.2. These measures can then be applied to the analysis of PUFs.

2.1 Basic Combinatorial Coverage and Coverage Difference Measures

Combinatorial methods offer an approach to coverage measurement that provides a measure directly related to fault detection. A series of studies have shown that most software bugs and failures are caused by one or two parameters and progressively fewer by three or more [19][20][21][22][5][6]. This finding means that testing parameter combinations can provide more efficient fault detection than conventional methods. This section, derived from [13], reviews the concept of measuring the combinatorial coverage of an input space [1][2][4] for use in testing or in other applications where it is important to ensure the inclusion of combinations of input parameter values.

	a	b	c	d
1	0	0	0	0
2	0	1	1	0
3	1	0	0	1
4	0	1	1	1

Figure 1. Example test array for a system with four binary components

Combinatorial coverage measurement concepts can be illustrated using the example in Figure 1, which shows a test array that contains 19 of a possible set of 24 2-tuples of values. To facilitate discussion, it is helpful to establish terminology for two related but distinct concepts:

- *t*-way *combination*: a set of *t* parameters or variables. For example, using the parameters in Figure 1, (*b*,*d*) is a 2-way combination, (*a*,*c*) is a different 2-way combination, and (*a*,*c*,*d*) is a 3-way combination.
- *t-tuple of values*: a combination for which the parameters have specific values. (Note: in the original definition from [1], this is referred to as a variable-value combination.) For example, (*b*=0, *d*=0) is one *t*-tuple of values, and (*b*=1, *d*=0) is a different *t*-tuple of values for the same 2-way parameter combination.

A simple combinatorial coverage of *t*-way combinations, S_t, is the fraction of possible *t*-tuples of values included in a test array from a domain D_t that may include constraints. With no constraints, where *v* is the number of values and *k* is the number of parameters, the size of the domain is $v^t\binom{k}{t}$ but may be smaller with constraints. For a set of *t*-tuples of values A_t in a test array,

$$S_t = \frac{|A_t|}{|D_t|}$$

Example: Figure 1 contains 19 different 2-way combinations out of a possible domain of $2^2\binom{4}{2}$, = 24 *t*-tuples of values, so S_t = 19/24 = 0.79.

Combinatorial coverage differences have been applied to several problem domains. Initially, this approach was used in fault identification, specifically to determine the particular combination(s) of parameter values that would trigger a fault. Another example problem where there is a need to distinguish one class of elements from another is anomaly-based intrusion detection, which seeks to determine if a particular exchange of packets represents an attempted network intrusion. Thus, it is useful to generalize the approach to find combinations that are present in one class or set and absent or rare in another, as well as to distinguish one set from another.

For fault location, if A_t = the set of *t*-tuples of values from passing tests and B_t = the set of *t*-tuples of values from failing tests, then the set difference $B_t \backslash A_t$ is of interest. These are the combinations in failing tests but not in passing tests, and thus, those that triggered a failure are contained in this set difference [26].

Example: If test #2 from Figure 1 is a failing test, then $B_t \backslash A_t$ = {bc = 10, cd = 10} is to be investigated to identify failing combinations because the four other 2-way *t*-tuples of values in test #2 are also contained in the passing tests #1, #3, #4, which are set A_t.

For transfer learning, if A_t = the set of *t*-way *t*-tuples of values from a source set of class instances and B_t = the set of t-tuples of values from a target set of instances, then the size of the set difference $B_t \backslash A_t$ as a fraction of the target set size is of interest as a metric of how similar the source is to the target set [27]. This set difference of t-tuples of values is: $\frac{|B_t \backslash A_t|}{|B_t|}$

2.2 Distinguishing Combinations

For many machine learning applications, the goal is to develop a model that distinguishes members of one class from another using attributes that identify them, such as distinguishing dogs from cats using attributes like size, ear shape, or hair texture. This publication will refer to sets being distinguished as either *Class* or *Non-class* sets. The terms Class and Non-class are used as generic terms for sets of objects that can be distinguished based on some attributes or properties. In a machine learning context, these sets may refer to concepts that are to be learned, such as distinguishing one animal species from others. In earlier applications, set differences of t-tuples of values have been used to identify the causes of failures [4][5]. In both cases, the process is the same – set differencing is used to identify combinations that occur in the class set that do not occur, or are rare, in the non-class set. If this difference is computed on t-tuples of values in failed tests versus passed tests, then the difference contains t-tuples of values that have triggered the failure (in a deterministic system). In machine learning, the difference represents properties or attributes that occur in the class (e.g., a particular animal species) that do not occur, or are rare, in the non-class examples (other species). Note that this is simply a generalized version of the original fault location problem, where the class whose distinguishing features are to be identified is the set of failing tests, and the features to be found are the combinations that lead to a test resulting in a failure.

The combinatorial coverage measures described in the previous section – as applied in fault location, explainability, and transfer learning – are based on the presence or absence of t-tuples of values in input files for testing or machine learning training. That is, a combination is counted as covered if it occurs once or multiple times in the input file, and this measure is appropriate in the applications discussed. For these applications, it is important to determine if a t-tuple of values has been included, but the number of times it occurs is less important. For testing, multiple occurrences of a combination mean some duplication of effort but do not affect the requirement for ensuring that all t-way combinations have been covered. In transfer learning evaluation, the same type of requirement holds – assurance that states and environments, as represented by t-tuples of values of the input model, are handled correctly. If it can be shown that the ML model produces the right prediction or classification for a t-tuple of values, multiple occurrences of the combination are not needed. (This does not consider the effect of input sequences; other measures are appropriate for sequence coverage.)

In other types of evaluations related to machine learning, it will be important to consider the number or frequency of occurrence of t-way t-tuples of values to determine the degree to which an attribute is associated with a particular class. If a particular combination of attribute values is seen in a high proportion of class members but not in non-class members, then it may be a reasonable indicator for distinguishing instances or at least for narrowing the range of possibilities for class identification. For example, many dog breeds may have a long tail, and many may have a curled tail, but a much smaller number of breeds have both attributes. Thus, it is important to have a measure that considers the quantity of instances with t-tuples of values in class and non-class instances.

This paper will abbreviate C_t and N_t as C and N, where interaction level t is clear or is not needed for discussion. The following discussion defines a t-way combination c_t as a distinguishing combination for the class C if it is present in a class instance of class set C and absent in non-class instances N, or if it is more common in C than N as determined by a threshold value. Two ways to identify distinguishing combinations are suggested below, and others are clearly possible. The key point is to use combinations of attribute values that are *strongly associated* with one class but not with others based on the frequency or rate of occurrence in one class as compared with others.

At least two possible ways to define the strength of association of a t-tuple of values with a class can be considered. These are defined and presented below as CFD1 and CFD2. (In a previous publication, only CFD1 was given as the definition of this strength of association [13].) The threshold T in definition CFD1 determines if a t-tuple of values c_t is common in set C_t and rare in set N_t and, thus, distinguishes one set from the other. Specifically, the definition below identifies t-tuples of values for which one can say "*x is T times more common in C than it is in N*" – an intuitive way to identify t-tuples of values that are associated closely with the class C. Note that the phrase "T times more common" suggests that T will normally be 1 or greater. For definition CFD2, U designates the threshold value. T may be any positive number, and U ranges from 0.0 to 1.0. Notice that these definitions produce the same result for inclusion or exclusion in the set of distinguishing combinations when $T = \frac{1}{1-U}$, or $U = \frac{T-1}{T}$. For example, if $T = 4$ or $U = 0.75$, then for pairs [($f(x_t, C_t)$; $f(x_t, N_t)$)], [.81; .2], and [.79; .2], the first will be found to be distinguishing, and the second will not.

CFD1 *Definition*: A combination x_t is *distinguishing* for a class $C \Leftrightarrow f(x_i, C_t) > T \times f(x_i, N_t)$, where $f(x_i, Y_t)$ = frequency of t-tuple of values x in set of t-tuples of values Y. The frequency f is the number of times a t-tuple of values appears in rows of the class over the number of rows for the class.

CFD2 *Definition*: A combination x_t is *distinguishing* for a class $C \Leftrightarrow \frac{f(x_i,C_t) - f(x_i,N_t)}{f(x_i,C_t)} > U$, where $f(x_i, Y_t)$ = frequency of t-tuple of values x in set of t-tuples of values Y. Note that, in this case, the threshold U ranges from 0.0 to 1.0. The frequency f is the number of times a t-tuple of values appears in rows of the class over the number of rows for the class.

The choice of CFD1 or CFD2 as a definition may depend on which is more intuitive for the application. Specifying $T = 1$ or $U = 0$ means that a combination is selected as distinguishing whenever it occurs at a higher frequency in C than N, no matter how small the difference in frequency.

2.3 Combination Frequency Difference Measures

The frequency (or rate) of occurrence refers to the number of times a t-tuple of values is present per number of rows in the file or array. Therefore, the combination frequency difference, for a t-tuple of values x in two arrays of instances of two different classes can be defined as the difference between the fraction of occurrences in one array and the second. That is, using the symbols defined below, CFD = $F_{Cx} - F_{Nx}$, where

R = number of rows of challenge-response file
R_C = rows of class instances; for PUFs, $R_C = R_1$ (i.e., where challenges produce a 1 response)
R_N = rows of non-class instances; for PUFs, $R_N = R_0$
k = number of columns or attributes, excluding class or response variable; for PUFs, $k = 64$
v = number of values for attributes; for PUFs, $v = 2$ as the attributes correspond to bits
M_{Cx} = number of occurrences of a particular t-tuple of values x in C
M_{Nx} = number of occurrences of a particular t-tuple of values x in N
$F_{Cx} = M_{Cx}/R_C$ = fraction of occurrences of a t-tuple of values in C
$F_{Nx} = M_{Nx}/R_N$ = fraction of occurrences of a t-tuple of values in N

The frequency difference values can be graphed, where the height on the Y axis shows the difference $F_{Cx} - F_{Nx}$ for every t-tuple of values x. The X axis is indexed by $v^t \binom{k}{t}$, points for t-way combinations. Thus, for each t-way combination, there are v^t possible values or settings of the t attributes or variables in the combination. For example, 2-way t-tuples of values are displayed in the order given by: *i,j for i in* $0 \le i < k-1$ *for j in* $i+1 \le j < k$. Thus, there are k-1 iterations of the inner loop on j for each attribute i, and for each 2-way combination, the graph displays the fraction of occurrences of each set of v^2 t-tuples of values on the X axis at $v^2\big((k-1)i+j-1\big)$ through $v^2\big((k-1)i+j-1\big)+v^2$-1. For each of these 2-way combinations x, F_{Cx} - F_{Nx} for four t-tuples of values are displayed for the four possible value settings 00, 01, 10, 11. Thus, in Figure 2, the difference in coverage for C and N for i =1, j = 4 will be found on the horizontal axis at x = 32..35.

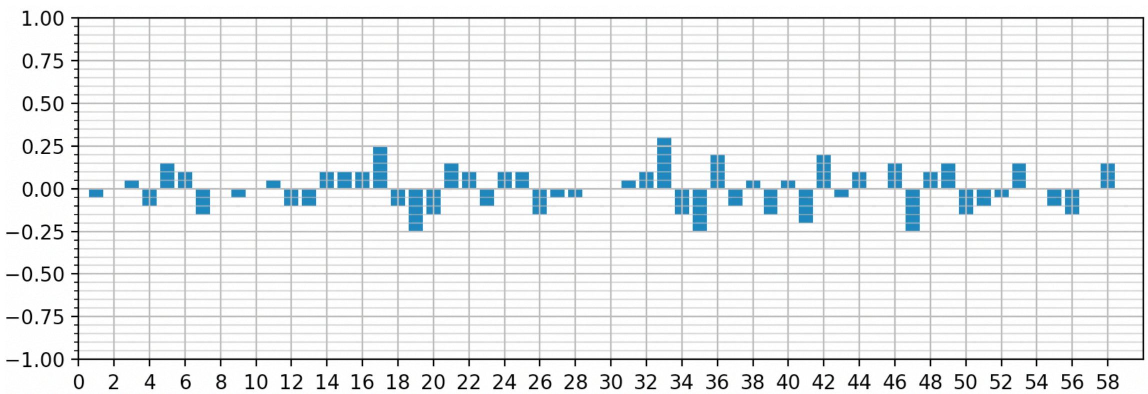


Figure 2. Example combinatorial frequency difference for two classes of 6 binary variables

For example, with n = 6 numbered 0..5, 2-way combinations will be indexed on the Y axis as (0,1,00), (0,1,01),…, (4,5,11), for a total of $2^2\binom{6}{2}$, = 60 X-axis points, numbered 0..59. For each of these, the Y-axis shows the difference in frequency of occurrence between C and N, normalized for the size of sets C and N. For example, if the value 01 for attribute combination i=1, j=4 occurs 40 times in a C file of 100 rows and 60 times in an N file of 120 rows, then the Y axis value for i,j = 1,4 for value 01 is (40/100) – (60/120) = -0.1. The analysis of PUFs described in this paper can use these quantities to identify bits related to internal structure.

3 Application to Physical Unclonable Functions

A physical unclonable function, or PUF, may be regarded as a physical implementation of a black box function that produces a response r for a given challenge string of bits c, that is, $r = f(c)$. The unit response is binary and can be represented as 0 or 1. A series of PUFs can be put together to produce a larger response sequence. As the name suggests, PUFs are designed using physical hardware devices. These functions utilize unique properties of the physical elements within the hardware, such as the small variation in propagation delays between identical circuit gates or small threshold mismatches in a transistor feedback loop due to process variation. These physical characteristics are difficult to reproduce in the hardware, which is what makes them physically unclonable. Using such physical characteristics, PUFs can be utilized to combat insecure storage, hardware counterfeiting, and other security problems.

An ideal PUF should be stable over time, unique in its existence, easy to evaluate, and difficult or impossible to predict. Thus, it should not be possible to generate a function that has the same behavior or produces the same output as the PUF for challenge inputs. In this sense, the PUF function is "unclonable." It should also be infeasible to determine components of the PUF that influence the output of the PUF, such that a 0 or 1 value in some positions of the input string makes a 0 or 1 output more likely for the output r.

The primary use of PUFs is related to authentication. In a simple use case, the physical system is subjected to one or more challenges during manufacturing, and the responses to these challenges are recorded. Later, if one of those recorded challenges is repeated and if the expected response is received, then the device is authenticated.

Depending on the strength of their implementation and consequent scalability, PUFs are categorized into two levels – weak and strong. Weak PUFs have a limited number of challenge-response pairs (CRPs) that can be generated from a single device, while strong PUFs can generate a much larger set of CRPs. One of the key requirements for a strong PUF design is that it should not be possible to infer information about the internal structure by observing inputs and outputs [16]. Many authors have shown that machine learning models can be constructed to predict the output of PUFs for a given input string (i.e., "breaking" the PUF by defeating its authentication function). Vulnerability to breaking through machine learning attacks can vary significantly with PUF design, and one of the challenges in developing PUFs is to identify potential weaknesses before constructing the PUF.

Table 1 shows ML prediction results for the five PUF designs discussed in this paper and for 10 ML algorithms available through the Weka machine learning tool package [17]. Note that ZeroR is a baseline, where predictions are simply the proportion of 0 or 1 results for the challenge/response pairs in the training set. The other algorithms were selected to provide a representative sample of popular ML algorithms of different types. AdaBoost is an adaptive ensemble algorithm that uses a phased sequence of basic decision tree algorithms, improving on prediction results with each phase. Bayes Net and Naïve Bayes are based on Bayesian statistical concepts. Decision Table is a majority classifier based on a nearest neighbor algorithm. J48 and Random Forest are based on decision trees. Stochastic gradient descent minimizes a loss function that is a weighted linear combination of the attributes, and logistic regression uses weighted attributes in a regression function. JRip is a propositional logic-based rule learning algorithm. Although there is a wide range of results for different algorithms, it is clear that DB1 – the arbiter design – is much more vulnerable to ML attacks, where two algorithms are able to predict the response to challenges with near perfect accuracy. Even the best two PUF implementations (DB3 and DB4) are not fully resistant to revealing some bias in their responses. Note that their averages are all considerably above the baseline ZeroR, which simply guesses in proportion to 0 or 1 responses in challenge-response pairs.

Table 1. ML Prediction results for five PUF designs

	Ada Boost	BayesNet	Decision Table	J48/C45	JRip	Logistic	Naïve Bayes	Random Forest	Stoc Grad Descent	ZeroR	Average accuracy	combined diff 2-way
DB1	77.1	96.2	75.6	72.1	77.2	**99.7**	96.2	87.2	99.3	55.0	86.7	0.489
DB2	54.8	54.9	**76.7**	68.1	75.2	54.9	54.9	71.9	52.4	55.6	62.6	0.309
DB3	50.7	50.1	**71.0**	63.9	67.2	50.3	50.1	62.6	50.2	50.1	57.3	0.248
DB4	57.5	56.5	58.8	54.6	**60.7**	56.4	56.5	55.3	54.6	50.6	56.8	0.216
NN00	64.1	64.8	62.1	59.1	64.8	64.8	64.8	**65.4**	62.6	50.5	63.6	0.383

This section shows how combination frequency differences of PUF input data can be used to determine a good deal of information about the design and internal structure of a PUF. This is achieved by measuring the difference between occurrences of *t*-way combinations associated with a 0 response as compared with a 1 response. Ideally, there should be little difference, except for random variances. As shown below, however, these differences vary considerably and align with the differences in predictability using machine learning. Although this work is only preliminary, this information may be useful in identifying design deficiencies and making PUFs more resistant to breakthrough machine learning.

Comparing the accuracy of ML predictions in Table 1 with the graphs in Figures 3 through 7, it is immediately apparent that there is a relationship between the "noisiness" of the graphs and the success of ML algorithms in predicting or breaking the PUF. The arbiter PUF, DB1 (Figure 3), response has a very noisy graph with differences for nearly every 2-way combination of bits ranging from about 0.10 to 0.25. For this PUF, ML algorithms predict the response with up to 99.7 % accuracy. For the PUF most resistant to ML predictions, DB4 (Figure 6), the graph shows small frequency differences with nearly all under 0.05 and up to a few around 0.10. The others fall within the range between DB1 and DB4 for both frequency differences and prediction accuracy, which is a metric for the potential of breaking the PUF. Maximum frequency differences for DB3 are around 0.12, for DB2 about 0.15, and for the neural net PUF around 0.19 – roughly consistent with the rankings of best, worst, and average for prediction accuracy and, hence, vulnerability to ML attacks. See the last column of Table 1, which shows the range for 2-way frequency differences above and below the center line, or $max(|f(x_i, C_t) - f(x_i, N_t)|) + max(|f(x_i, N_t) - f(x_i, C_t)|)$.

There are two major types of hardware implementation of PUFs: memory-based and delay-based. A typical memory-based PUF is the SRAM PUF. Delay-based PUFs include arbiter PUFs, the pseudo-linear feedback shift register PUF, and the ring oscillator (RO) PUF.

3.1 Arbiter PUF (DB1)

The main idea of an arbiter PUF is to create a digital race for signals through two paths within a chip and to have an *arbiter* circuit that decides which signal has won the race. The two paths are designed identically. However, the manufacturing process usually introduces a very slight longer delay in one of the paths from the other. Given a particular challenge, the arbiter PUF will therefore produce an output dictated by the physical characteristics of that unique hardware implementation. During an arbiter PUF design, one has to make sure that the delays between the two paths are not too close to each other. Otherwise, the output will be dictated by noise in the signal rather than the delay uniquely introduced through the manufacturing variation.

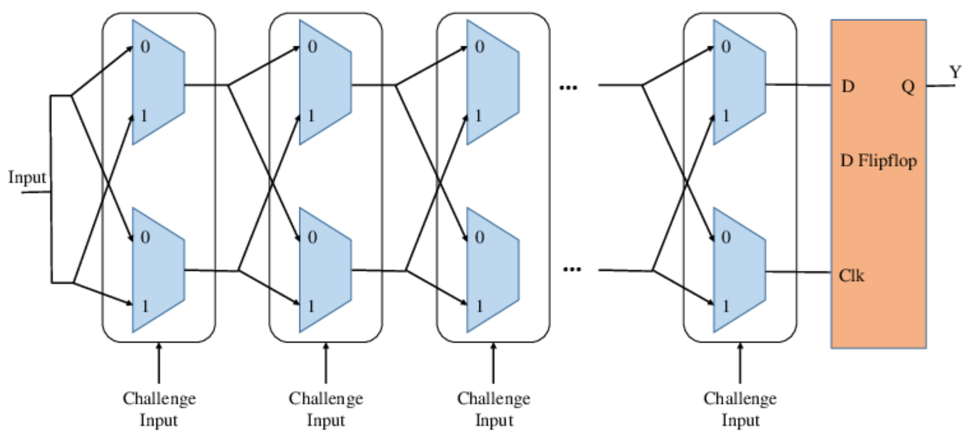


Figure 3. Basic operations of an arbiter PUF

As Figure 3 shows, each gate or switch-block introduces a delay for one of the outputs, which accumulates over the blocks. This gives rise to the opportunity of building what is typically known as *model-based attacks* (also known as *model building attacks* or *model learning attacks*). The idea is that one can build a mathematical model of the PUF which, after observing several CRP queries, will be able to predict the response for a given challenge with a high level of accuracy. With the proliferation of machine learning algorithms, this type of model building or model learning has become easier to implement. To make model building attacks more difficult on basic arbiter PUFs, non-linearity is introduced into the delay lines of the designed circuit. For example, in case of feed-forward arbiter PUFs, some challenge bits are not set by the user. Rather, they are connected to the outputs of the intermediate arbiters evaluating the race at some intermediate point the circuit. This technique, however, increases the noise in the output of the arbiter PUF. Although initial results with feed-forward arbiter PUFs were shown to be resistant to model-building or model-learning attacks, more sophisticated learning models were able to break them [17].

By simply analyzing combination frequency differences (CFD) within a subset of the challenge-response pairs (CRPs) and without knowing anything about the type or design of the circuitry, one can predict which arbiter PUF design is likely to be more vulnerable to model-building attacks.

Figure 3(a) shows 2-way frequency differences for a 64-bit PUF, DB1, an early arbiter design with delays placed randomly in the hardware. With 64 bits, there are $2^2\binom{64}{2}$ = 8064 2-way differences indexed. Differences range from a low of -0.23881 to a high of 0.25108 for a range of 0.48990. Note that differences are given as difference F_{Cx} - F_{Nx}., so negative values are cases where non-class t-tuples of values exceed class t-tuples of values.

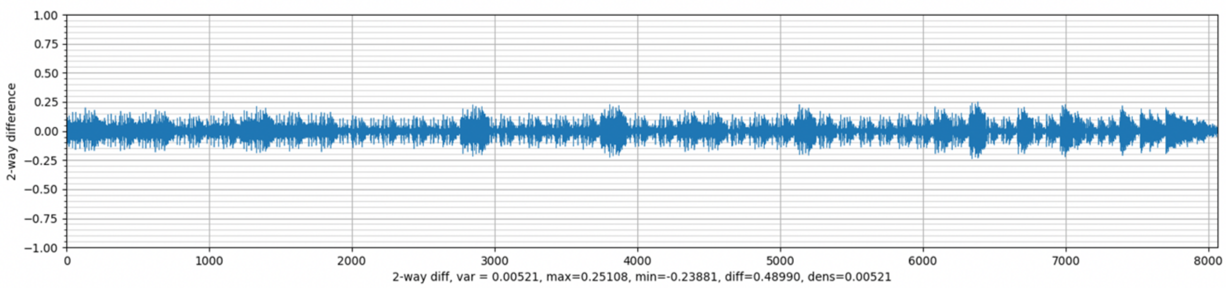


Figure 3(a). 2-way frequency differences for a 64 bit arbiter PUF

Figure 3(b) shows 3-way frequency differences for the same PUF. Note that variance, minimum, and maximum differences are smaller than those for 2-way combinations. The X axis indexes $2^3\binom{64}{3}$, = 333,312 combinations.

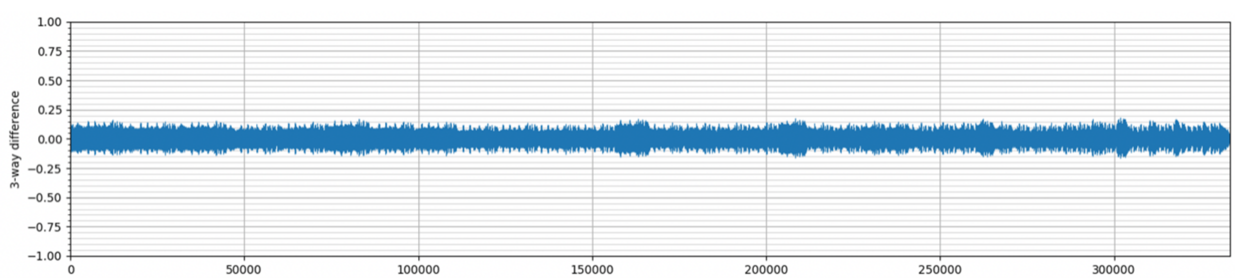


Figure 3(b). 3-way frequency differences for a 64 bit arbiter PUF

3.2 8-bit Shift Register PUF (DB2)

Shift register PUF is another delay-based PUF implementation, where a series of linear feedback shift registers

(LFSR) are put together to capture the unique delays associated with a physical implementation. Researchers have proposed pseudo-LFSR-based physically unclonable functions, known as PL-PUF, which are usually small in size, efficient in producing authentication ID for devices, and easy to modify to adjust the challenge-response pairs when needed [29].

This section examines the security of a shift register-based PUF against a model-based attack using combinatorial frequency difference analysis. Frequency differences for an 8-bit shift register type of PUF are shown in Figure 4(a) (2-way) and Figure 4(b) (3-way). Note that the variance is much smaller – 0.00017 compared to 0.0521 for 2-way combinations of DB1 inputs. There is much more uniformity in the response of DB2 to 2-way and 3-way combinations of input bits, and as expected, this makes it much more difficult for ML to derive a model for the PUF that can successfully reproduce its response to inputs.

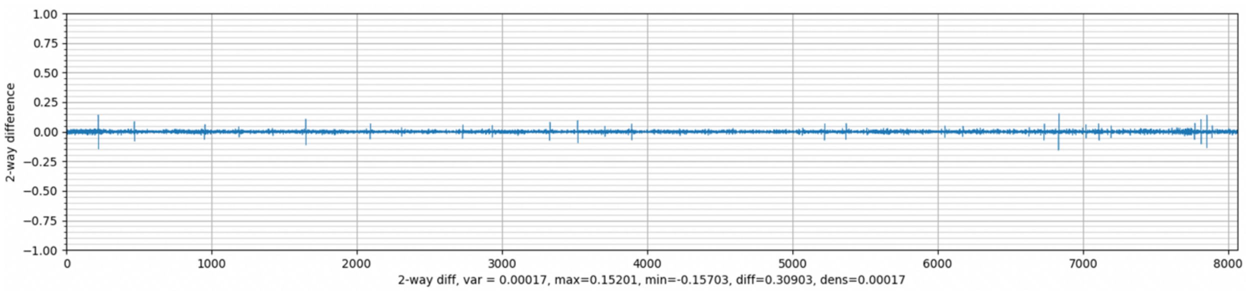


Figure 4(a). 2-way frequency differences for an 8-bit shift register PUF

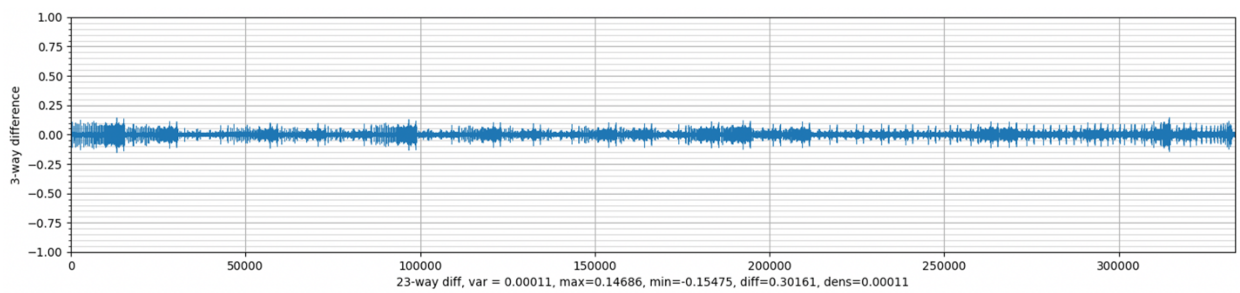


Figure 4(b). 3-way frequency differences for an 8-bit shift register PUF

However, Figure 3(a) also shows a small number of spikes in the combination frequency chart. Combinations producing these spikes are shown in Table 2, which shows 2-way bit combinations where the frequency difference exceeds 3σ. Combinations of almost all bits with bit 56 result in a spike that exceeds 3σ (others have spikes that are slightly below this value but still clearly different from the other combinations). The appearance of spikes compresses towards the right end of the graph because combinations are indexed in a loop computation: *i,j,b: for i in* $0 \le i < 63$ *for j in* $i+1 \le j < 64$ *for b in* {00,01,10,11}, similarly for 3-way combination indexes.

Table 2. 2-way combinations with greatest frequency differences in Figure 4(a)

bits = values	bits = values	bits = values	bits = values
(0,56) = (1,0)	(11,56) = (0,0)	(26,56) = (1,1)	(41,56) = (0,1)
(0,56) = (0,1)	(12,56) = (1,0)	(26,56) = (0,0)	(42,56) = (1,1)
(1,56) = (1,0)	(12,56) = (0,1)	(31,56) = (1,1)	(42,56) = (0,0)
(1,56) = (0,1)	(14,56) = (1,1)	(31,56) = (0,0)	(51,56) = (1,1)
(3,56) = (1,1)	(14,56) = (0,0)	(32,56) = (1,1)	(51,56) = (0,0)
(3,56) = (0,0)	(15,56) = (1,0)	(37,56) = (1,1)	(52,56) = (1,0)
(4,56) = (1,0)	(15,56) = (0,1)	(37,56) = (0,0)	(52,56) = (0,1)
(6,56) = (1,0)	(16,56) = (0,1)	(38,56) = (1,1)	(53,56) = (1,1)
(6,56) = (0,1)	(17,56) = (1,0)	(38,56) = (0,0)	(53,56) = (0,0)
(8,56) = (1,1)	(17,56) = (0,1)	(40,56) = (1,0)	(54,56) = (1,1)
(8,56) = (0,0)	(25,56) = (1,1)	(40,56) = (0,1)	(54,56) = (0,0)
(11,56) = (1,1)	(25,56) = (0,0)	(41,56) = (1,0)	

A potential explanation can be developed for the pattern of spikes in combinations that include bit 56 by noting that 8 is an even divisor of 56. PUFs accumulate differences as steps progress, so bit 56 occurs at the final stage before the last 8-bit shift register. In a design situation, the next step would be to analyze the hardware components to determine why this irregularity was occurring.

3.3 32-bit Shift Register PUF (DB3)

This section shows the results of the analysis performed on a 32-bit shift register PUF. As the name suggests, a 32-bit shift register PUF is designed the same as an 8-bit shift register, where the circuitry is four times longer. The added circuitry increases the complexity of the PUF and, thus, likely makes it a little less susceptible to model-building attacks.

The results of applying the analyses are shown in Figures 5(a) and 5(b).

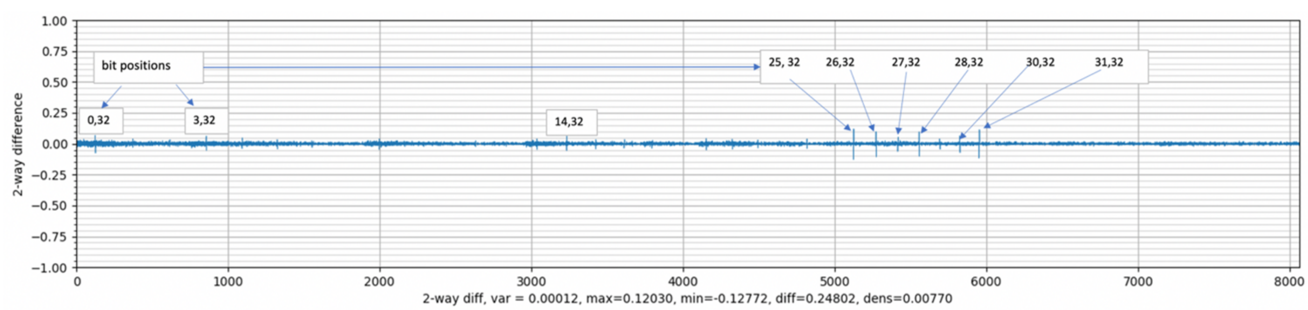


Figure 5(a). 2-way frequency differences for a 32-bit shift register PUF

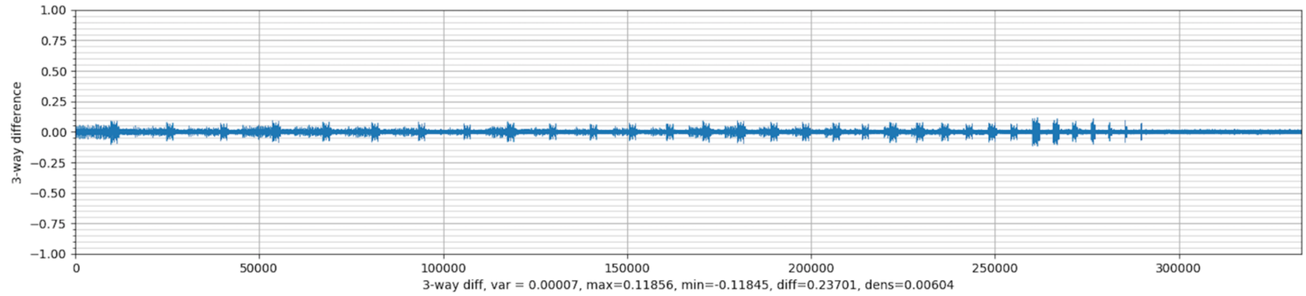


Figure 5(b). 3-way frequency differences for a 32-bit shift register PUF

3.4 Uniform distribution PUF (DB4)

Figure 6 shows results for a PUF with the most uniform distribution of all studied here. This PUF has the greatest resistance to machine learning attacks, which are able to predict responses only somewhat better than chance (see Table 1). In this case, the variations used in producing PUF responses accumulate uniformly with slight frequency differences for *t*-tuples of bits that include either bit 61 or 62. (Compression of the spikes towards the right side of the graph occurs because of the loop computation, as explained in Section 3.2.)

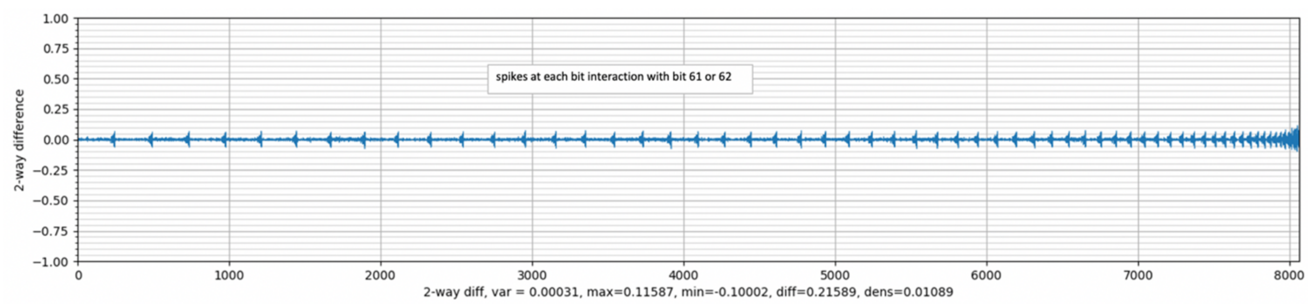


Figure 6(a). 2-way frequency differences for a uniform distribution PUF

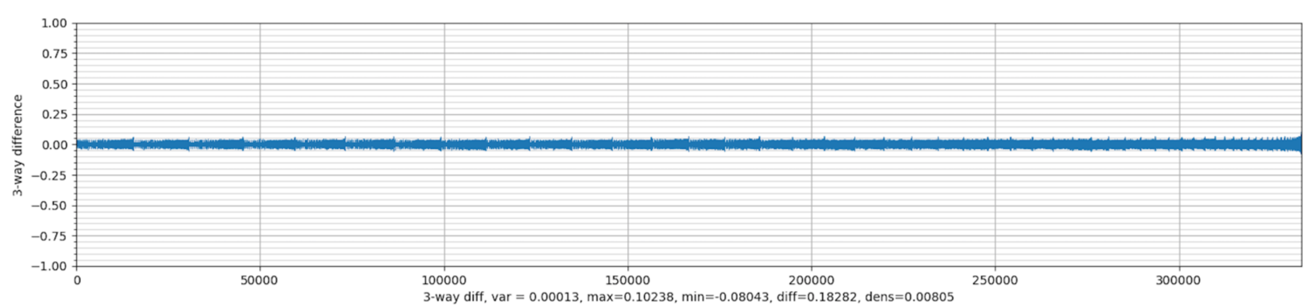


Figure 6(b). 3-way frequency differences for a uniform distribution PUF

3.5 Neural Net PUF

Researchers have pointed out the vulnerabilities of arbiter and other types of PUFs, especially against model-building attacks [30]. To thwart the model-learning attack, researchers proposed both a simple neural network (NN) [31] as well as recurrent neural network (RNN)-based PUFs [32]. These new models are specifically designed for high resistance to model-building attacks achieved by introducing non-linearity between the challenge-response pairs. The physical implementation uses current-mirrors to construct the PUF. The basic idea is to propagate a current through two identical chains of non-linear current mirrors. In the case of RNN-based PUF, the circuitry feeds back the challenge bits into the PUF. [32]

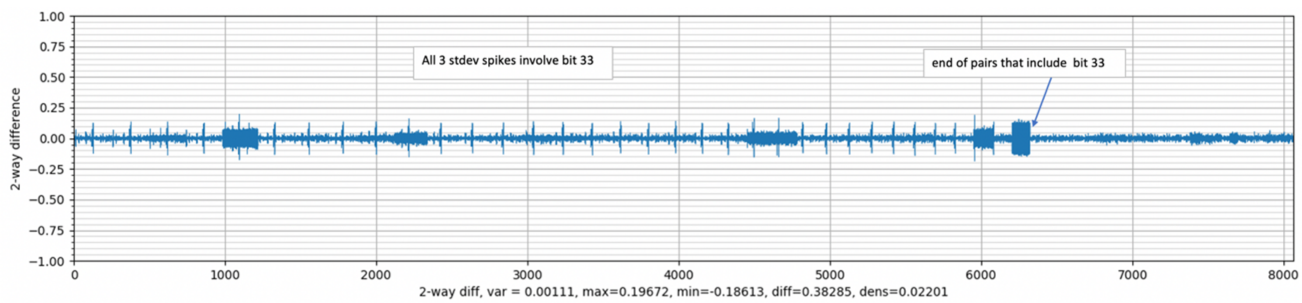


Figure 7(a). 2-way frequency differences for a neural net PUF

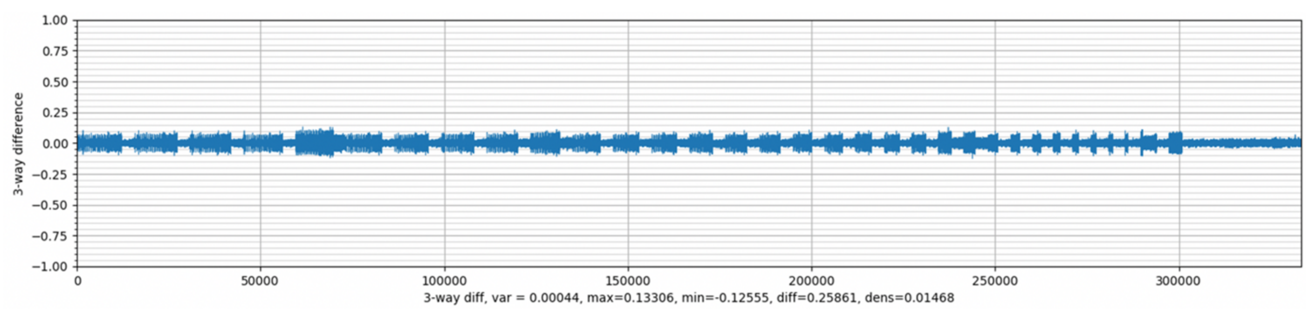


Figure 7(b). 3-way frequency differences for a neural net PUF

4 Extension to Machine Learning

A *distinguishing combination* has been defined as one present in a class instance of class set *C* and absent in non-class instances *N*, or if it is more strongly associated with *C* than *N*, as determined by a threshold value. As the name suggests, a distinguishing combination is one that differentiates one type or class of instance from others. Thus, it is natural to consider if these combinations can be used directly in machine learning problems for predicting class membership from instance attributes. If an instance contains many *t*-tuples of values that are associated with a particular class but not with other classes, then it is likely to be a member of the class with which the *t*-tuples of values are strongly associated. This section shows that initial results suggest this approach works quite well in many cases. No ML algorithm is best for all problems, and the CFD approach to classification performs better than other ML algorithms for some problems and less well for others. This section reviews some of these empirical results and suggests future work to characterize the conditions under which CFD machine learning will be advantageous.

Given a set of distinguishing combinations, a simple algorithm for classification seems natural: if an instance has more attribute combinations that are associated with a class *C* than another class, then assign it to *C*, and if there are fewer combinations associated with *C* than another class, then assign it to the other class. (For simplicity, only two classes are considered here, but the method can be extended to multiple classes by considering each one as "*C*" in turn). If the *C* and *N* combinations are equally present, then the result is undermined. As the saying goes, "if it looks like a duck and walks like a duck and quacks like a duck (a 3-way combination), it's probably a duck!"

CFD algorithm:

```
dist_c = {distinguishing combinations for instances in class C}
dist_n = {distinguishing combinations for instances not in class C}

dc = sum(1 for t-way combinations x_i in row if x_i in dist_c)
dn = sum(1 for t-way combinations x_i in row if x_i in dist_n)
if dc > dn:  predict C
if dn > dc:  predict N
if dc == dn: indeterminate
```

A number of possible alternatives to the basic algorithm can be conceived. Perhaps the most obvious is to weigh the presence of distinguishing combinations in instances, shown below as CFDw. Using a weight of $|F_{Cx} - F_{Nx}|$, the CFDw algorithm has been compared with the basic CFD for several examples. Comparisons of the weighted method with the basic method are shown in the following sections along with frequency difference graphs. Accuracy scores for CFD and CFDw are relatively close, and there is no clear winner between these two variations.

CFDw algorithm:

```
dc = sum(weight(x_i) for t-way combinations x_i in row if x_i in dist_c)
dn = sum(weight(x_i) for t-way combinations x_i in row if x_i in dist_n)
if dc > dn:  predict C
if dn > dc:  predict N
if dc == dn: indeterminate
```

Using this approach on the PUF data presented in the previous section produces results that are relatively comparable to the ML algorithms shown in Table 1 for 10,000 rows using 4-way combinations shown in Table 3.

Table 3. Comparison of CFD accuracy with average, best, worst from Table 1

	CFD	Avg, Table 1	Best, Table 1	Worst, Table 1
DB1	.953	86.7	.997 (logistic)	.721 (J48)
DB2	.547	62.6	.767 (dec tbl)	.524 (SGD)
DB3	.520	57.3	.710 (dec tbl)	.501 (Bayesnet)
DB4	.546	56.8	.607 (JRip)	.546 (SGD)
NN00	.621	63.6	.654 (Rand Forest)	.591 (J48)

As previously discussed, PUFs are designed to be “unclonable” (i.e., difficult to replicate, including through strategies such as machine learning). In most ML applications, the classes of interest are in nature or may be industrial products not designed to be resistant to modeling. This difference is also immediately apparent in the graphs in Appendix A, which show much wider variation for these “natural” or practical datasets. An example is shown in Figure 8 below (mushroom data set from Appendix):

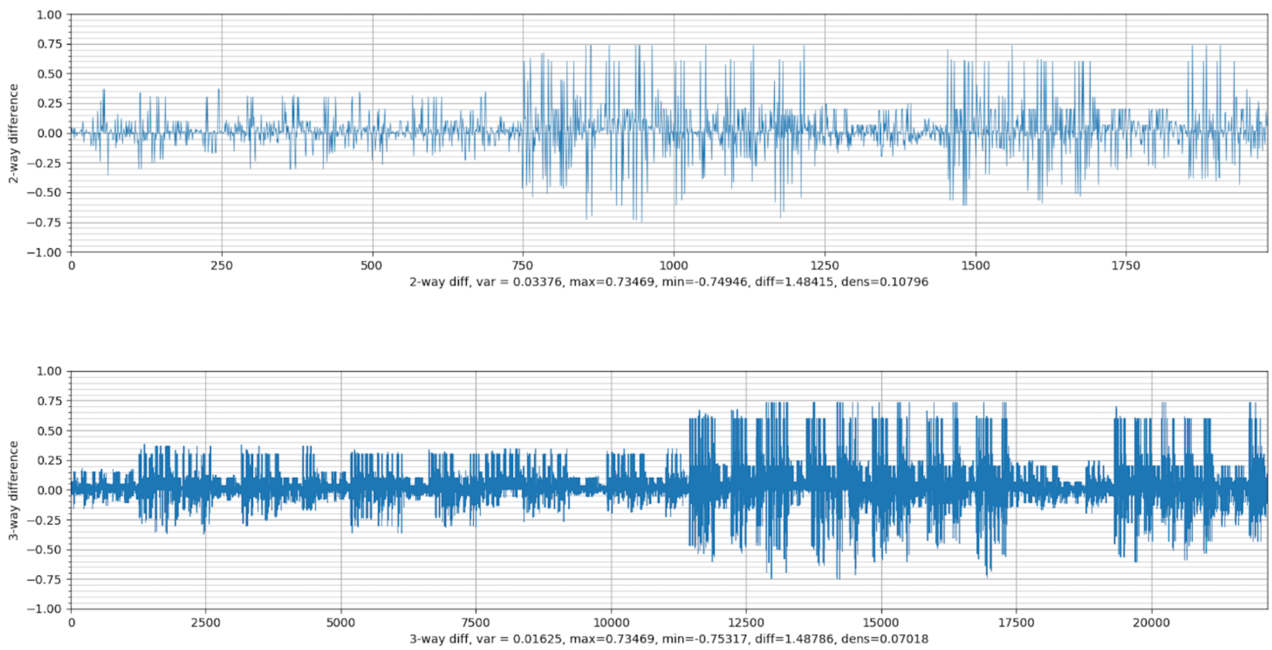


Figure 8. Frequency difference graph for 2-way and 3-way differences, mushroom example

As shown in this graph and others in Appendix A, there is a much wider variation in frequency differences – up to roughly 75 % or more. The much smaller variation for PUFs is likely due to the fact that they are designed to be difficult to clone or replicate. The wider range of frequency differences in these natural examples make the CFD approach more effective, using the differences to distinguish between classes. On these applications, CFD class prediction does quite well, as shown in Table 4. Accuracy scores in the column labeled “CFD4 @ T” are the average of 10 random assignments of the total number of rows given by “*n* rows” split into 66 % training and 34 % test for the threshold of T shown using 4-way combinations.

Table 4. Comparison of CFD accuracy with other ML algorithms

Dataset	n row	n col	n class	n non	CFD4 @ T	Ada	Baye	DecTbl	J48	JRip	Log	NB	Rand	SGD	ZR
Bcanc	286	9	68	218	.970@1.0	.759	.766	.745	.769	.720	.752	.752	.745	.766	.762
Coupon	12684	25	7210	5474	.730@5.0	.644	.663	.688	.718	.725	.693	.663	.757	.684	.569
Credit	1000	20	37	963	.991 @ 5.1	.963	.950	.962	.963	.957	.958	.949	.963	.963	.963
Diab	768	8	367	401	.992@1.0	.698	.723	.709	.694	.692	.728	.723	.674	.715	.522
Heart2	47786	21	23893	23893	.755@5.0	.745	.741	.745	.757	.754	.767	.741	.753	.762	.500
Mush	5644	22	2156	3488	1.00 @ 1.0	.963	.985	1.00	1.00	1.00	1.00	.974	1.00	1.00	.618
Soyb	684	31	133	551	.986 @ 15.0	.991	.968	.988	.981	.972	.975	.929	.983		.845

It is important to note that a small number of threshold values have been tried. Further experimentation with threshold values and characterization of their applicability will be the subject of future research. An additional issue to be investigated is the possibility of overfitting. Two of the sample machine learning data sets have less than 10 attributes. Using 4-way combinations to test for membership in class or non-class sets may have a potential for overfitting because a 4-way combination could include roughly half of the attributes available for classifying an instance. The other data sets were chosen with more than 20 attributes to reduce the possibility of overfitting. A detailed investigation of this issue will be the subject of future research.

5 Conclusions

This paper presents a method for measuring and visualizing differences in the frequency or rate of occurrence of t-way combinations for two data sets. This measure, combination frequency differencing (CFD), has potential use in a variety of applications. Initially applied to challenge-response pairs for physical unclonable functions of PUFs, CFD was shown to provide the ability to identify combinations of bits in the challenge that are more or less strongly associated with particular output values of 0 or 1. The level of difference appears to correlate with the effectiveness of machine learning attacks on PUFs. In future research, the authors hope to develop ways to trace these strongly non-uniform bit combination associations to the hardware components that produce them. This ability might be useful in the design and development of PUFs to identify design weaknesses and correct them before production.

It was also shown that the basic idea behind CFD can be extended to produce a new type of machine learning algorithm. CFD identifies and measures differences between two data sets using attribute value combinations, and this approach lends itself naturally to identifying instances in classification problems. An instance that is very similar to others of a particular class is likely to be a member of that class. This paper shows that the accuracy of this CFD approach to classification problems is comparable to the accuracy of well-known algorithms across a variety of problem types. Further research is planned to investigate developing this method into a practical approach for classification problems. In previous work, the authors have used the concept of unique or distinguishing combinations for explainability in AI/ML systems [23][28], so there may be effective methods for combining the CFD method for classification with explainability.

References

[1] Kuhn, D. R., Mendoza, I. D., Kacker, R. N., & Lei, Y. (2013, March). Combinatorial coverage measurement concepts and applications. In *2013 IEEE Sixth International Conference on Software Testing, Verification and Validation Workshops* (pp. 352-361). IEEE.

[2] Mendoza, I. D., Kuhn, D. R., Kacker, R. N., & Lei, Y. (2013, October). CCM: A tool for measuring combinatorial coverage of system state space. In *2013 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement* (pp. 291-291). IEEE.

[3] Kuhn, D.R., Kacker, R.N. and Lei, Y., 2010. Practical combinatorial testing. *NIST special Publication*, *800*(142), p.142.

[4] Kuhn, D. R., Kacker, R. N., & Lei, Y. (2015). Combinatorial coverage as an aspect of test quality. *CrossTalk*, *28*(2), 19-23.

[5] Z. Ratliff, R.Kuhn, R. Kacker, Y.Lei, K. Trivedi, The Relationship Between Software Bug Type and Number of Factors Involved in Failures, submitted to International Workshop Combinatorial Testing, 2016.

[6] Li, X., Gao, R., Wong, W.E., Yang, C. and Li, D., Applying combinatorial testing in industrial settings. In *2016 IEEE Intl Conf on Software Quality, Reliability and Security (QRS)* (pp. 53-60).

[7] Fifo, M., Enoiu, E., & Afzal, W. (2019, April). On measuring combinatorial coverage of manually created test cases for industrial software. In *2019 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)* (pp. 264-267). IEEE.

[8] Smith, R., Jarman, D., Bellows, J., Kuhn, R., Kacker, R., & Simos, D. (2019, April). Measuring Combinatorial Coverage at Adobe. In *2019 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)* (pp. 194-197). IEEE.

[9] Mayo, Q., Michaels, R., & Bryce, R. (2014, March). Test suite reduction by combinatorial-based coverage of event sequences. In *2014 IEEE Seventh International Conference on Software Testing, Verification and Validation Workshops*(pp. 128-132). IEEE.

[10] Morgan, J. (2018). Combinatorial testing: an approach to systems and software testing based on covering arrays. *Analytic Methods in Systems and Software Testing*, 131-158.

[11] Ozcan, M. (2017, March). Applications of practical combinatorial testing methods at siemens industry inc., building technologies division. In *2017 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)* (pp. 208-215). IEEE.

[12] Chandrasekaran, J., Lei, Y., Kacker, R., & Kuhn, D. R. (2021, April). A Combinatorial Approach to Explaining Image Classifiers. In *2021 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)* (pp. 35-43). IEEE.

[13] Kuhn, R., Raunak, M. S., & Kacker, R. (2021). *Combinatorial Coverage Difference Measurement (Draft)*. National Institute of Standards and Technology.

[14] Vijayakumar, A., Patil, V. C., Prado, C. B., & Kundu, S. (2016, May). Machine learning resistant strong PUF: Possible or a pipe dream? In *2016 IEEE international symposium on hardware oriented security and trust (HOST)* (pp. 19-24). IEEE.

[15] Ghandehari, L. S., Chandrasekaran, J., Lei, Y., Kacker, R., & Kuhn, D. R. (2015, April). BEN: A Combinatorial Testing-based Fault Localization Tool. In Software Testing, Verification and Validation Workshops (ICSTW), 2015 IEEE Eighth International Conference on (pp. 1-4). IEEE.

[16] Herder, C., Yu, M. D., Koushanfar, F., & Devadas, S. (2014). Physical unclonable functions and applications: A tutorial. *Proceedings of the IEEE*, *102*(8), 1126-1141.

[17] Witten, I. H., & Frank, E. (2002). Data mining: practical machine learning tools and techniques with Java implementations. *Acm Sigmod Record*, *31*(1), 76-77.

[18] Majzoobi, M., Koushanfar, F., Potkonjak, M.: Testing techniques for hardware security. In: Test Conference, 2008. ITC 2008. IEEE International, pp. 1{10 (2008)

[19] D.R. Kuhn, D.R. Wallace, A.M. Gallo, Jr., Software Fault Interactions and Implications for Software Testing, IEEE Transactions on Software Engineering, vol. 30, no. 6, June 2004, pp. 418-421. Comment: Investigates interaction level required to trigger faults in a large distributed database system.

[20] D.R. Kuhn and M.J. Reilly, An Investigation of the Applicability of Design of Experiments to Software Testing, 27th Annual NASA Goddard/IEEE Software Engineering Workshop (SEW ’02), Greenbelt, Maryland, December 5-6, 2002, pp. 91-95

[21] D. R. Kuhn, V. Okun, Pseudo-exhaustive Testing for Software, 30th Annual IEEE/NASA Software Engineering Workshop (SEW-30), Columbia, Maryland, April 24-28, 2006, pp. 153-158

[22] Cotroneo, D., Pietrantuono, R., Russo, S., & Trivedi, K. (2016). How do bugs surface? A comprehensive study on the characteristics of software bugs manifestation. *J.Systems and Software*, *113*, 27-43.

[23] R. Kuhn, R. Kacker, An Application of Combinatorial Methods for Explainability in Artificial Intelligence and Machine Learning. *NIST Cybersecurity Whitepaper*, May 22, 2019.

[24] DR Kuhn, R Kacker, Y Lei, D Simos, "Combinatorial Methods for Explainable AI", *Intl Workshop on Combinatorial Testing*, Porto, Portugal, March 23-27, 2020.

[25] Simos, D. E., Kleine, K., Voyiatzis, A. G., Kuhn, R., & Kacker, R. (2016, August). Tls cipher suites recommendations: A combinatorial coverage measurement approach. In *2016 IEEE International Conference on Software Quality, Reliability and Security (QRS)* (pp. 69-73). IEEE.

[26] Laleh Sh Ghandehari, Yu Lei, Raghu Kacker, Richard Kuhn, Tao Xie, and David Kung. A combinatorial testing-based approach to fault localization. IEEE Transactions on Software Engineering, 46(6):616–645, 2018.

[27] Lanus, E., Freeman, L. J., Kuhn, D. R., & Kacker, R. N. (2021, April). Combinatorial Testing Metrics for Machine Learning. In *2021 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)* (pp. 81-84). IEEE.

[28] Kampel, L., Simos, D. E., Kuhn, D. R., & Kacker, R. N. (2021). An exploration of combinatorial testing-based approaches to fault localization for explainable AI. *Annals of Mathematics and Artificial Intelligence*, 1-14.

[29] Y. Hori, H. Kang, T. Katashita and A. Satoh, Pseudo-LFSR PUF: A Compact, Efficient and Reliable Physical Unclonable Function, *2011 International Conference on Reconfigurable Computing and FPGAs*, 2011, pp. 223-228, doi: 10.1109/ReConFig.2011.72.

[30] U. Ruhrmair, F. Sehnke, J. Solter, G. Dror, S. Devadas, and J. Schmidhuber, Modeling Attacks on Physical Unclonable Functions. *2010. Proc. of the 17th ACM Conference on Computer and Communications. Security (CCS '10) ACM. 237-249.* https://doi.org/10.1145/1866307.1866335

[31] R. Kumar and W. Burleson. 2014. On design of a highly secure PUF based on non-linear current mirrors. In 2014 IEEE International Symposium on HardwareOriented Security and Trust (HOST). 38ś43. https://doi.org/10.1109/HST.2014.6855565

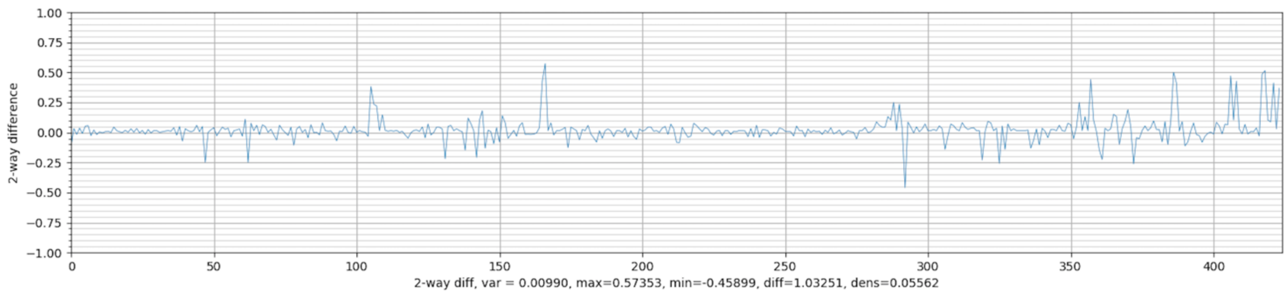
[32] Shah, Nimesh, et al. "A 0.16 pj/bit recurrent neural network based PUF for enhanced machine learning attack resistance." *Proceedings of the 24th Asia and South Pacific Design Automation Conference*. 2019.

Appendix A—Difference Graphs of Classification Problems

This section presents examples of typical machine learning classification problems taken from the UCI Machine Learning Repository (https://archive.ics.uci.edu/ml) or from Kaggle (https://www.kaggle.com). Each example includes the data source, associated publication, and results from the tools described in this paper.

Bcanc - https://archive.ics.uci.edu/ml/datasets/Breast+Cancer

Michalski,R.S., Mozetic,I., Hong,J., & Lavrac,N. (1986). The Multi-Purpose Incremental Learning System AQ15 and its Testing Application to Three Medical Domains. *Proceedings of the Fifth National Conference on Artificial Intelligence*, 1041-1045, Philadelphia, PA: Morgan Kaufmann.



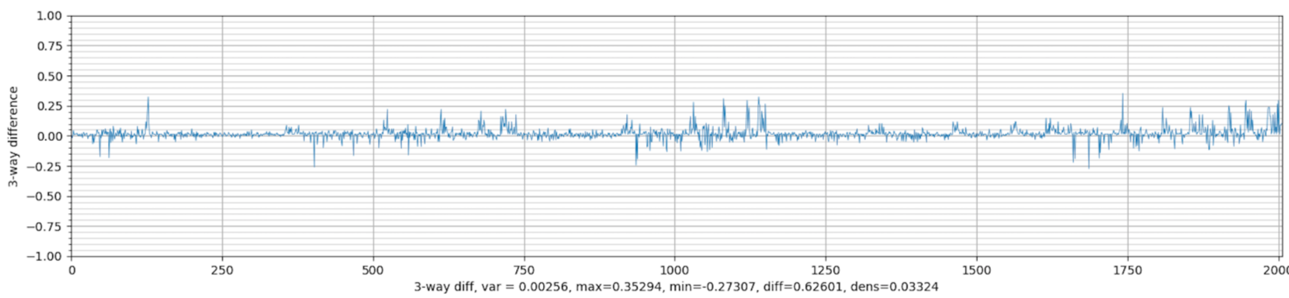


Figure 9. *Breast cancer data frequency differences.*

```
CFD results:
== confusion matrix 4-way ==
  |      C    |       N    <- predicted
C |     75    |       0
N |      3    |      21
=====================
Accuracy:  0.970


CFDw results:
== confusion matrix 4-way ==
  |      C    |       N    <- predicted
C |     24    |       0
N |      2    |      73
=====================
Accuracy:  0.980
============================
```


Coupon - https://www.kaggle.com/mathurinache/invehicle-coupon-recommendation

Wang, Tong, Cynthia Rudin, Finale Doshi-Velez, Yimin Liu, Erica Klampfl, and Perry MacNeille. 'A Bayesian framework for learning rule sets for interpretable classification.' The Journal of Machine Learning Research 18, no. 1 (2017): 2357-2393.

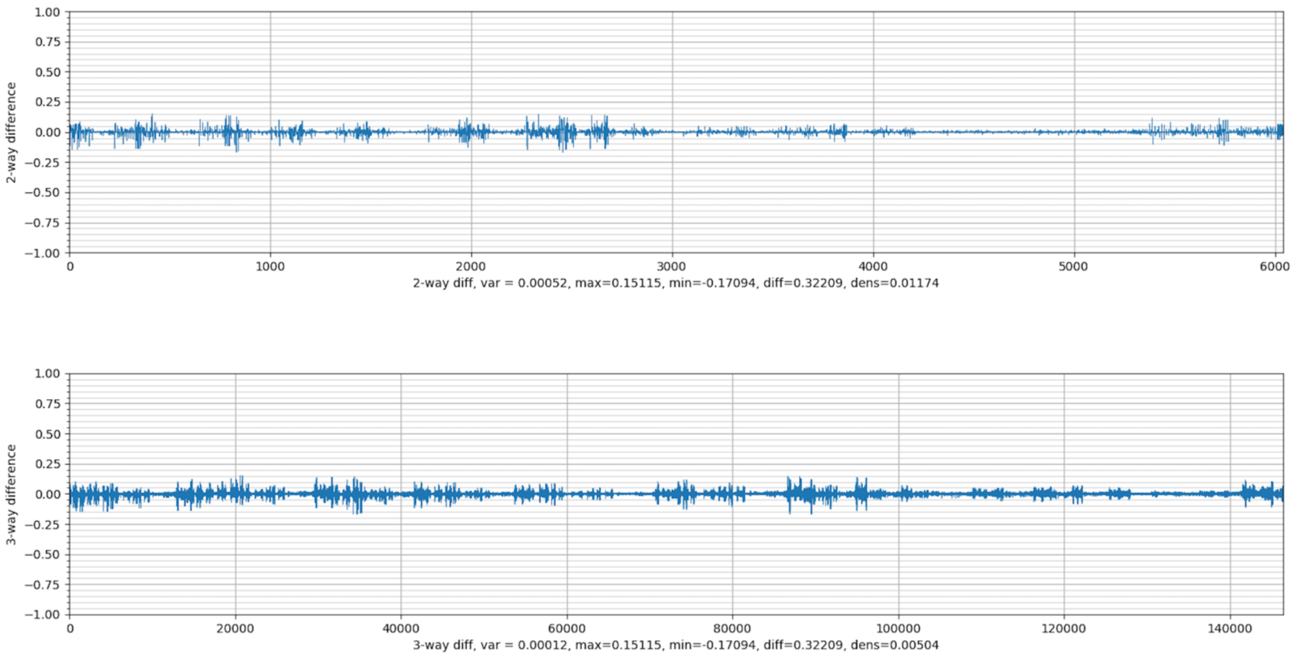
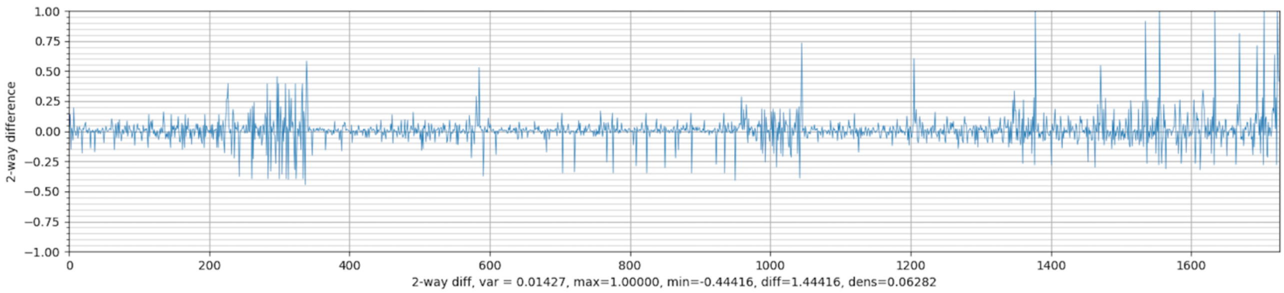


Figure 10. *Coupon data frequency differences.*

```
== confusion matrix 4-way ==
  |      C   |       N    <- predicted
C |   1931   |     521
N |    661   |    1201
=====================
Accuracy:  0.726
============================
```

Credit - https://archive.ics.uci.edu/ml/citation_policy.html
Dua, D. and Graff, C. (2019). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science.



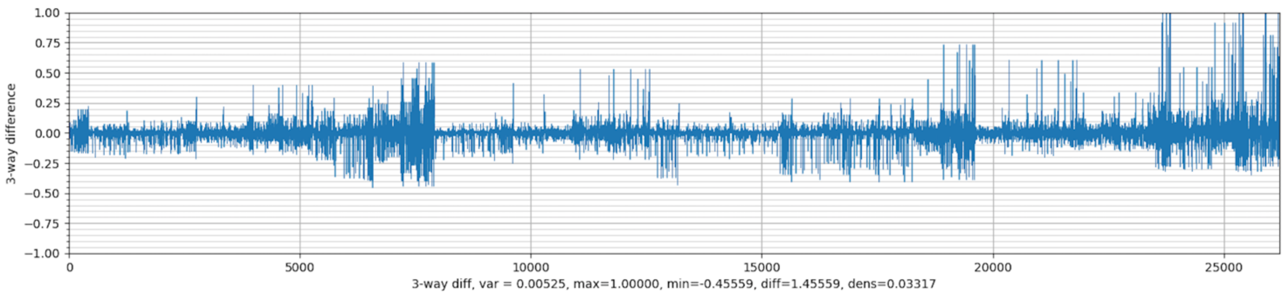


Figure 11. *German credit check data frequency differences.*

```
== confusion matrix 4-way ==
  |      C    |       N    <- predicted
C |     10    |       3
N |      0    |     328
=====================
Accuracy:  0.991
============================


CFDw results:
== confusion matrix 4-way ==
  |      C    |       N    <- predicted
C |    293    |      35
N |      0    |      13
=====================
Accuracy:  0.897
============================
```

Diab - https://archive.ics.uci.edu/ml/datasets/diabetes

Smith, J.W., Everhart, J.E., Dickson, W.C., Knowler, W.C., Johannes, R.S. (1988). Using the ADAP learning algorithm to forecast the onset of diabetes mellitus. *Proceedings of the Symposium on Computer Applications and Medical Care* (pp. 261--265). IEEE.

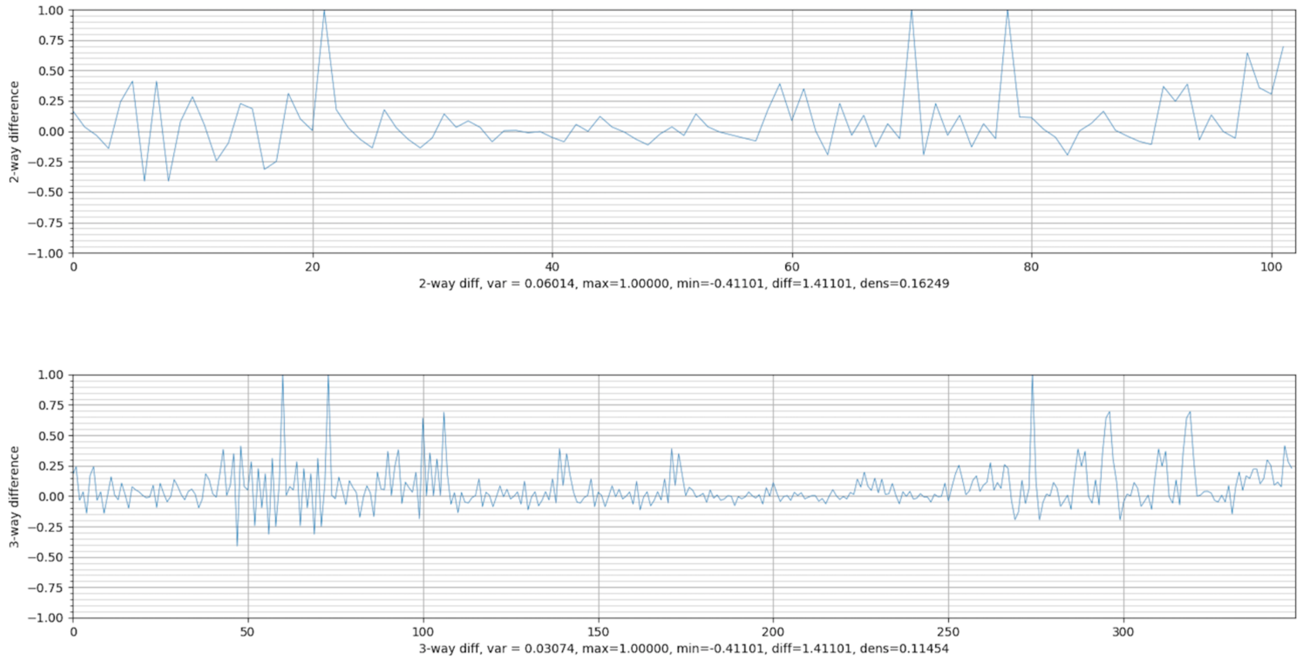


Figure 12. ***Diabetes data frequency differences.***

```
CFD results:
== confusion matrix 4-way ==
  |      C    |       N    <- predicted
C |    124    |       1
N |      1    |     136
=====================
Accuracy:  0.992

CFDw results:
== confusion matrix 4-way ==
  |      C    |       N    <- predicted
C |    125    |       0
N |      1    |     136
=====================
Accuracy:  0.996
============================
```

Heart2 - https://github.com/doguilmak/Heart-Diseaseor-Attack-Classification

Large set of data containing 253,681 instances, with 23,893 heart disease or attack, and the rest healthy. To make instance sets equal size, a random set of 23,893 disease/attack instances were extracted.

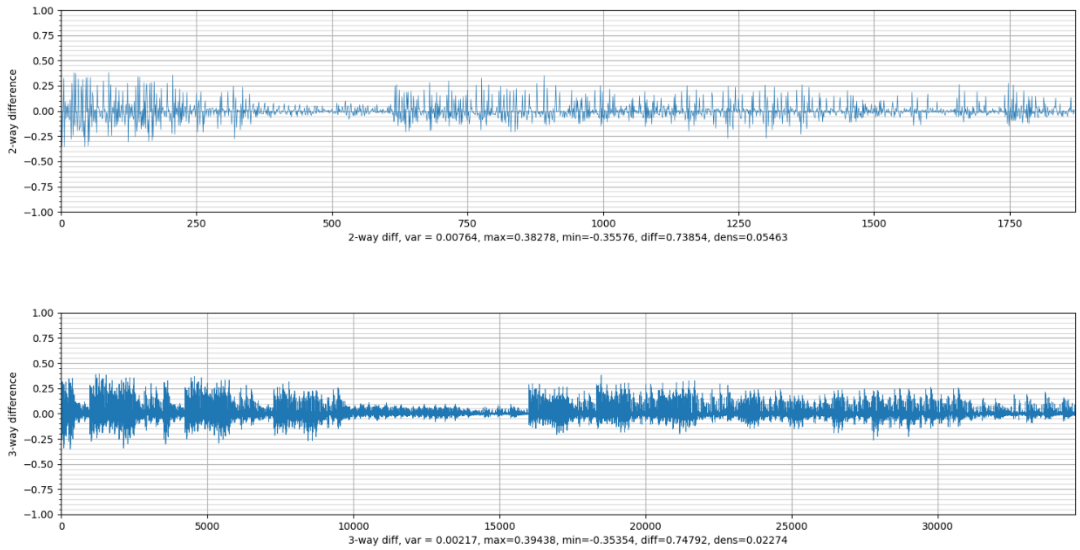


Figure 13. ***Heart disease data frequency differences.***

```
== confusion matrix 2-way ==
  |      C    |       N    <- predicted
C |   6254    |    1870
N |   2472    |    5652
=====================
Accuracy 2-way = 0.733, SD_2 = 0.08709, dt = 1.046

== confusion matrix 3-way ==
  |      C    |       N    <- predicted
C |   6017    |    2107
N |   1950    |    6174
=====================
Accuracy 3-way = 0.750, SD_3 = 0.04643, dt = 1.046
== confusion matrix 4-way ==
  |      C    |       N    <- predicted
C |   5905    |    2219
N |   1764    |    6360
=====================
Accuracy 4-way = 0.755, dt = 1.046

== confusion matrix 5-way ==
  |      C    |       N    <- predicted
C |   5859    |    2265
N |   1991    |    6133
=====================
Accuracy 5-way = 0.738, dt = 1.046
=====================
```

Mush - https://archive.ics.uci.edu/ml/datasets/Mushroom

Schlimmer, J.S. (1987). Concept Acquisition Through Representational Adjustment (Technical Report 87-19). Doctoral dissertation, Department of Information and Computer Science, University of California, Irvine.

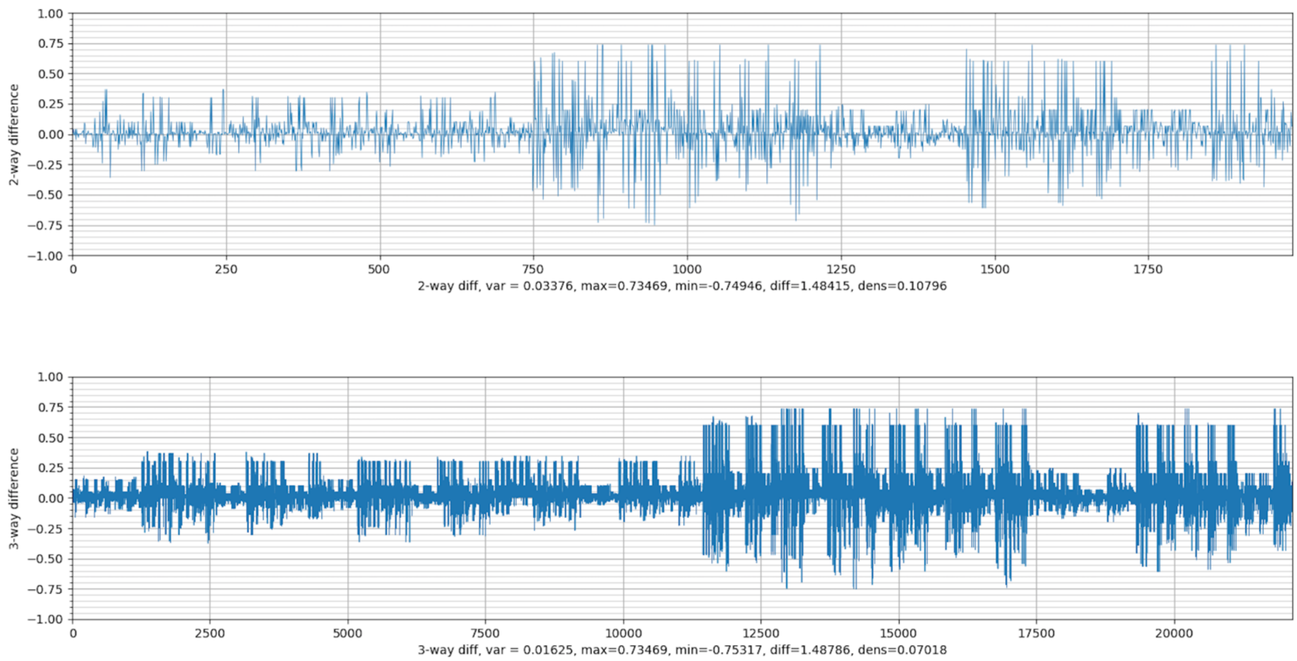


Figure 14. ***Edible mushroom data frequency differences.***

```
CFD results:
== confusion matrix 4-way ==
  |      C    |        N    <- predicted
C |    725    |        9
N |     58    |     1128
=====================
Accuracy:  0.965


CFDw results:
== confusion matrix 4-way ==
  |      C    |        N    <- predicted
C |    553    |      181
N |      0    |     1186
=====================
Accuracy:  0.906
============================
```

Soyb - https://archive.ics.uci.edu/ml/datasets/Soybean+%28Large%29

R.S. Michalski and R.L. Chilausky. "Learning by Being Told and Learning from Examples: An Experimental Comparison of the Two Methods of Knowledge Acquisition in the Context of Developing an Expert System for Soybean Disease Diagnosis", *International Journal of Policy Analysis and Information Systems*, Vol. 4, No. 2, 1980.

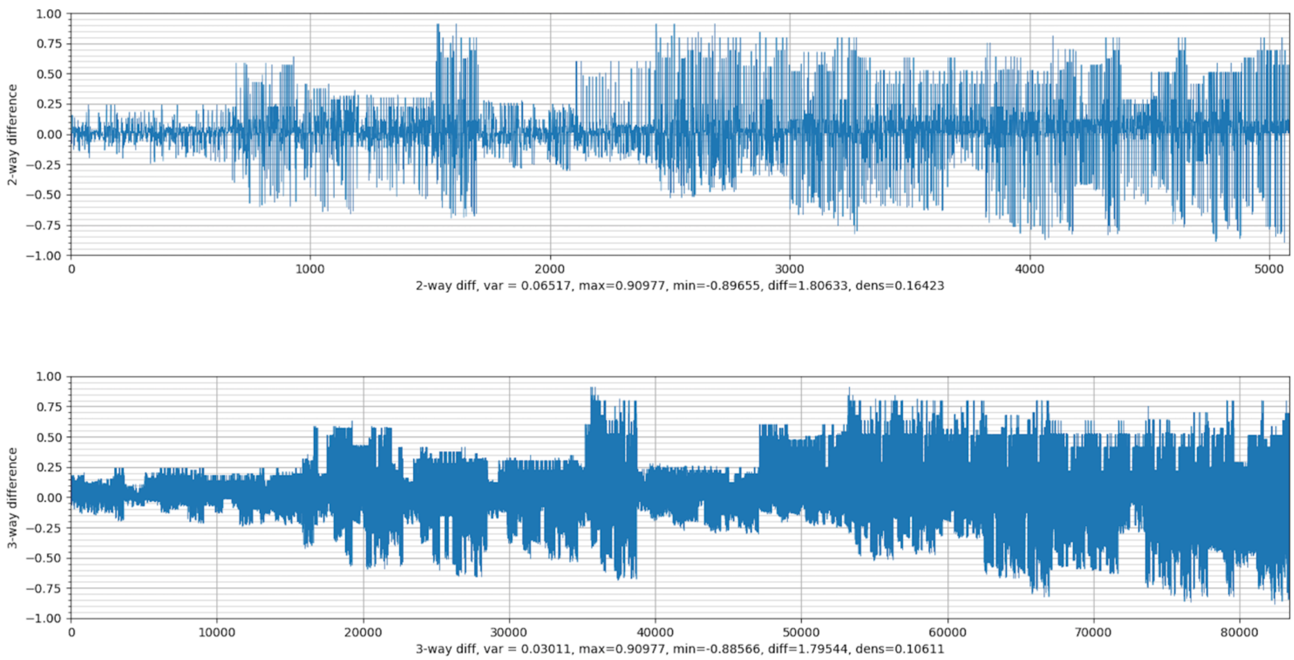


Figure 15. *Soybean disease data frequency differences.*

```
CFD results:
== confusion matrix 4-way ==
  |      C    |        N    <- predicted
C |     42    |        4
N |      0    |      188
=====================
Accuracy:  0.983


CFDw results:
== confusion matrix 4-way ==
  |      C    |        N    <- predicted
C |     41    |        5
N |      0    |      188
=====================
Accuracy:  0.979
============================
```