

Pulse-type classification for the Large Underground Xenon dark matter search

Kelly Stifter

I. INTRODUCTION

The Large Underground Xenon (LUX) detector [1] is a particle physics experiment that is searching for dark matter. The main component of the detector is a large vessel filled with liquid xenon that acts a target for dark matter particles. An incoming particle interacts with an atom of xenon and creates an initial burst of photons and liberates electrons. The electrons then drift upwards through the liquid xenon and release a second burst of photons as they cross the liquid-gas interface through a process called electroluminescence.

The first and second bursts of photons are called S1 and S2. These signals are detected by grids of photomultiplier tubes above and below the liquid xenon. Based on differences between S1 and S2 waveforms, it can be predicted whether the incoming particle interacted with the electron cloud or nucleus of a xenon atom. The key to detecting a dark matter signal is accurately separating these types of events.

Therefore, it is imperative that S1 signals, S2 signals, and other pulse types are labeled correctly with high precision. The importance of this task is increased because of the vanishingly small proposed rate of dark matter interaction and the relatively high amount of background events. While the current classification performed by the LUX data-processing software is $\sim 99\%$ accurate, it has several limitations.

For these reasons, I developed a new pulse-type classifier which takes various quantities relating to a pulse waveform, and predicts a class for the pulse. My algorithm is able to match the current LUX classification in accuracy, while providing new insights into the physics of the processes being studied and a confidence level for each classification.

II. PREVIOUS WORK

The current algorithm used by the LUX data-processing software utilizes user-defined cuts to separate pulse populations. The pulses are plotted using various reduced pulse quantities (RQs), such as pulse area and pulse width, and population boundaries are drawn in various parameter spaces based on user-defined cuts defined by simulation of physics processes. This algorithm works well and is thought to have an accuracy approaching 99%. Its strength is separating populations that are distinct from one another in parameter space, but it is known to have issues in boundary regions.

Other low-background experiments perform event classification in similar ways. The Cryogenic Dark Matter Search (CDMS) defines a number of cuts in parameter space that are based on calibration data. Approximately half the calibration data is used to define the cuts, and the remaining half is used to calculate the cut efficiencies and estimate expected

backgrounds. It has been shown that the efficiency of these cuts ranges between 20-60% [3]. Though these cuts allow the CDMS collaboration to make very sensitive predictions on the existence of dark matter, it is clear that the efficiency of the cuts could be improved.

To this end, CDMS has investigated event classification through machine learning. Brown graduate Michael Attisha included his work on the application of neural networks to event analysis in his Ph.D. thesis [3]. His goal was to discriminate between two different types of events using a multilayer perceptron network. After accounting for overfitting through an empirical Bayes procedure, he obtained significant improvement ($\sim 40\%$) over the primary analysis of the same data. While this is a notable improvement, the technique is still limited by the fact that the labels for the training data were deduced through cuts, a process similar to the one defined above. Ideally, the labels would be produced in some other way, more independent of the physics.

The thought to improve the pulse classification process using machine learning has also recently come up within the LUX collaboration. UC Davis graduate student James Morad has started a working group for machine learning using LUX data, and pulse-type classification is the first goal. He has already implemented a nearest neighbors classifier using the hand-scan data described in Sec. III. Unfortunately, nearest neighbors will share shortcomings with the current LUX algorithms - the main populations will be well-defined, but the contested regions will suffer.

III. DATA SET

There are five main classes of pulses in this problem. The first two are S1 and S2, as described earlier, which are the signals that LUX is looking for. Therefore, it is very important to be able to separate these two pulse types from each other, and from the different types of noise. S1 signals are typically narrow pulses of medium intensity, while S2 signals are longer and of much higher amplitude.

The next two classes are single photons and single electrons, which are the most abundant classes. These two classes are produced by the same physics processes that produce S1 and S2 signals; they are just deemed too small to include in the final sample. This can make it difficult for classifiers to tell the difference, so single photons are often confused with S1 signals and single electrons with S2 signals.

The final main pulse-type is noise. This class is a catch-all for many things, including electronic noise and other physics processes that are not of interest. Examples of all pulse classes can be seen in Fig. 1, 2, and 3.

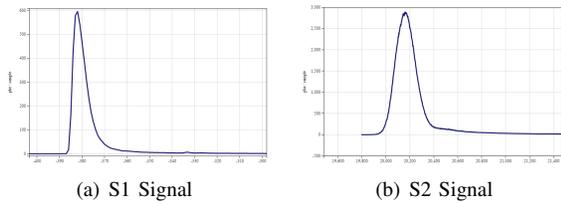


Fig. 1: *Left*: S1 signals are shorter and less intense than S2 signals. *Right*: S2 signals are longer and more intense than S1 signals.

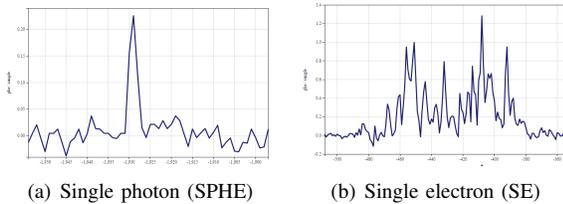


Fig. 2: *Left*: Single photons are small, single peaks produced through the same physics process as S1 signals. *Right*: Single electrons are long collections of small peaks produced through the same physics process as S2 signals.

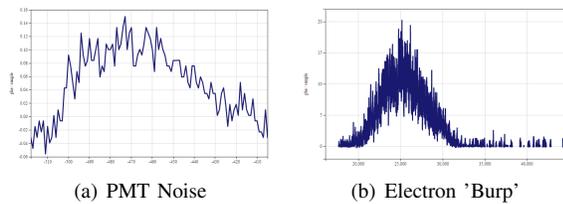


Fig. 3: Two examples of noise. *Left*: Electronics noise from a PMT. *Right*: A physics process called an 'electron burp'.

From each event in LUX data, there is a list of RQs that have been calculated by the LUX data-processing software. Fifty of these RQs are specific to individual pulses. They include things like pulse width, pulse area, etc. Of the 50 pulse RQs, 42 were viable for use as features, meaning they differed between samples and were never infinity. This subset of features was considered for use in the final classifier. Before use in training, all the features were normalized to have a mean of zero and a standard deviation of one.

There are two sources of existing labels for this data. The first is from the LUX data-processing software. The second is a hand-scan of pulses that was performed by various LUX scientists. Principle Component Analysis was performed on the data for visualization purposes, and can be seen in Fig. 4 use the LUX labels, and 5 using the hand-scan labels.

James Morad created a website [4] for the purpose of hand classification by LUX scientists. Users are shown a pulse and asked to select one of eleven classifications, one of the options being 'I don't know'. The remaining ten classes consist of the five main classes described above, as well as five sub-classes within the main classes. To date,

approximately 4000 pulses have been classified. The pulses classified as 'I don't know' were removed from the training data, so a total of 3652 training samples were used.

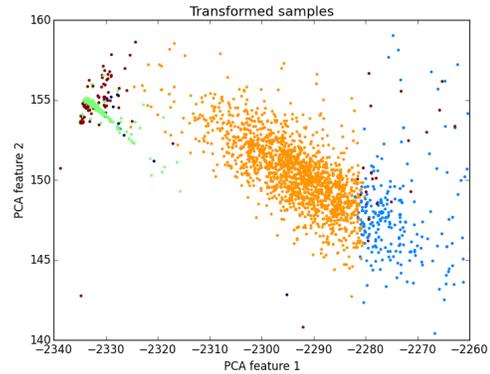


Fig. 4: Principle component Analysis performed on the LUX-classified data for the purposes of visualization. The various colors represent the 5 possible classifications for pulses. Red is S1, orange is S2, green is SPHE, light blue is SE, and dark blue is noise.

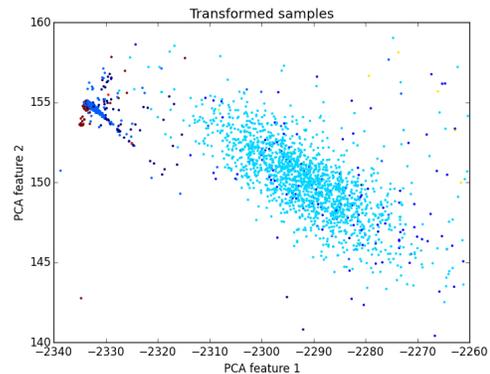


Fig. 5: Principle component Analysis performed on the hand-scan data for the purposes of visualization. The various colors represent the 11 possible classifications for pulses.

IV. METHODS

A number of classifiers were tested on this data set. Three multi-class classifiers were used: Naive Bayes (NB), support vector machines (SVMs), and random forests (RFs). I also defined two classifiers myself, which I denote Combined with Base (CwB) and Combined without Base (CwoB).

The first multi-class algorithm used was Naive Bayes, which estimates the probability of each class using Bayes rule:

$$P(y|x) = \frac{P(x|y)P(y)}{P(x)}$$

In the above equation, y is the label for the training sample and x is the feature vector. The naive assumption that all

features are independent leads to:

$$P(y|x) = \frac{P(y) \prod_{i=1}^n P(x_i|y)}{P(x)} \propto P(y) \prod_{i=1}^n P(x_i|y)$$

Finally, the label that maximizes this probability is chosen:

$$\hat{y} = \operatorname{argmax}_y P(y) \prod_{i=1}^n P(x_i|y)$$

The specific version of Naive Bayes that is implemented in the scikit-learn python package [5] assumed that the likelihood of each feature was Gaussian, which is quite close to the truth. With this, values for $P(Y)$ and $P(x_i|y)$ can be estimated by maximizing the joint likelihood of the data:

$$\mathcal{L}(\phi_{y,j}, \phi_{x,k,l}) = \prod_{i=1}^m P(x^{(i)}, y^{(i)}) = \prod_{i=1}^m P(x^{(i)}|y^{(i)})P(y^{(i)})$$

The second multi-class algorithm used was an SVM. SVMs define an optimal margin between two classes by solving a dual problem to maximize the distance between training samples and the classification boundary:

$$\begin{aligned} \max_{\alpha} W(\alpha) &= \sum_{i=1}^m \alpha_i - \frac{1}{2} \sum_{i,j=1}^m y^{(i)}y^{(j)} \alpha_i \alpha_j \langle x^{(i)}, x^{(j)} \rangle \\ \text{s.t. } 0 &\leq \alpha_i \leq C, \quad i = 1, 2, \dots, n \\ &\text{and } \sum_{i=1}^m \alpha_i y^{(i)} = 0 \end{aligned}$$

For the case of this problem, a Gaussian kernel was used, so the inner product in the equation above is replaced with:

$$\langle x^{(i)}, x^{(j)} \rangle = \exp(-\gamma |x^{(i)} - x^{(j)}|^2)$$

This algorithm provides a classification between two classes. In order to support a full multi-class classifier, a one-vs-one approach is taken. A separate classifier is built for each pair of classes, and the results from all classifiers are averaged.

The final multi-class algorithm used was a random forest [6]. The backbone of a random forest is a collection of decision trees. A tree is created by splitting the training set into subsets based on some criterion on the features that defines the 'best' splitting for that set. The process is then continued, and each subset is split into further subsets using the same criteria. Here, the criterion used to split sets was minimizing the Gini impurity. This is a measure of how likely a random sample from the subset is to be mislabeled, if it were randomly labeled based on the existing distribution of labels in the subset. Using the relative frequency of class j at node t , $P(j|t)$, the Gini impurity can be calculated as:

$$G(t) = 1 - \sum_j P(j|t)^2$$

The idea of a random forest is that each tree in the collection is trained not only on a randomly selected subset of training samples, but also on a randomly selected subset of features when splitting a node. The scikit-learn implementation combines the results from all trees by considering the probabilistic prediction for each class.

The CwB classifier is the first of two algorithms that I defined. This classifier takes five individual classifiers: one base multi-class classifier which classifies all data into all classes being used, and four binary classifiers (denoted 'experts') which perform a one-vs-rest classification by separating classes one through four from the rest of the data.

The result from the 'base' classifier is taken as the foundation of the labels. In most cases, only one of the experts will select a training example for its class, and the base classification is overwritten by the expert. In the case that two or more experts claim it, rules are introduced to break ties. If a sample is classified as both class one and class three, it is assigned to class one. Similarly, if a sample is assigned to class two and class four, it is assigned to class two. This decision is informed by the fact that it is worse to miss an S1 or S2 signal than it is to analyze an extra noise signal, and also by the fact that it was found in a series of small experiments that the correct label in these cases was always class one or two, and never class three or four. In all other ties (whether they are two-, three-, or four-way), the classifier defaults to the value given by the base classifier.

The CwoB classifier is very similar to the CwB classifier, except that it has no base classification. It takes only four binary classifiers to define the first four classes. Any sample that is not claimed by one of the four experts, or any tie that is not broken by the rules described by the CwB classifier, is instead simply classified as a fifth, noise class.

V. RESULTS

One way to quantify the success of a classifier is accuracy: the percentage of training samples that are labeled correctly. It would be unfair to train an algorithm on a set of data, and then test against that same set. For this reason, an extra step called cross-validation is taken when measuring accuracy.

Cross-validation involves splitting the data set into k subsets. For this problem, the data was split into four folds using a method called stratified cross-validation, in which the percentage of samples from each class is preserved in each subset. This was done because some of the classes have far fewer samples than others, and a subset that lacks a class is not ideal. The classifiers are then trained on three of the four folds of data, and tested on the remaining fold. This is done four times, holding out a different fold each time for testing.

Using the accuracy of the training and testing sets, a learning curve can be created. A learning curve plots the training error and the testing error as a function of the number of training samples.

A confusion matrix is a table that makes it clear if two classes are being confused with one another. The (i, j) element of the matrix represents the number of samples that are predicted to have label j , and are actually labeled i .

Precision and recall are two other common measures of success of a classifier. Precision is defined as the number of correctly predicted positives divided by the total number of predicted positives, and recall is defined as the number of correctly predicted positive divided by the total number of actual positives.

A. Results based on hand-scan data

The hand-scan data was first classified using several off-the-shelf classifiers, implemented in the scikit-learn package. Additionally, one of the classifiers that I defined, the CwB algorithm, was also tested. Both variations of this algorithm used RF experts, but one used an RF base while the other used an SVM with an RBF kernel as the base. These were implemented with default values for the hyper-parameters. The accuracy of each classifier, as computed by 4-fold stratified cross-validation, is shown in Table I.

Naive Bayes	0.61 ± 0.06
SVM	0.80 ± 0.01
RF	0.81 ± 0.01
CwB, RF base, RF experts	0.81 ± 0.01
CwB, SVM base, RF experts	0.81 ± 0.01

TABLE I: Accuracy of algorithms in classifying hand-scan data.

The algorithm with the highest accuracy was the CwB classifier with an SVM base and RF experts, so this one was selected for further investigation. A python package called Optunity [7] was used to perform hyper-parameter optimization on the RF experts to select values for the number of trees to create in each forest and the maximum number of features to consider at each node. Additionally, feature selection on the SVM base was performed, and it was found that while there are only four features that significantly contribute to the classification, the optimal number to use is seven. Even after these optimizations were implemented, the accuracy did not increase by a statistically significant amount. A sample confusion matrix is shown below.

$$\begin{bmatrix} 73 & 1 & 63 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 53 & 1 & 53 & 1 & 0 & 0 & 0 & 0 & 0 \\ 10 & 4 & 268 & 10 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 9 & 0 & 484 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 9 & 0 & 1 & 19 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 2 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 8 & 0 & 3 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 9 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 8 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Fig. 6: Confusion matrix for the CwB classifier with an SVM base and RF experts. Created by training the classifier on 60% of the training set, and testing on the remaining 40%.

As can be seen, classes 1 and 3 are often confused with one another, as are classes 2 and 4. The experts were supposed to address this issue, but even the experts can not distinguish a single class from the rest of the data. There are other problem areas, as can be expected with unbalanced classes. It is especially problematic since some of the smaller classes are technically sub-classes of one of the four main classes. For example, class 9 is a sub-class of class 1. Therefore it is

no surprise that all samples from class 9 are mis-classified as class 1.

More problems are evident when the learning curve, shown in Fig. 7, is plotted.

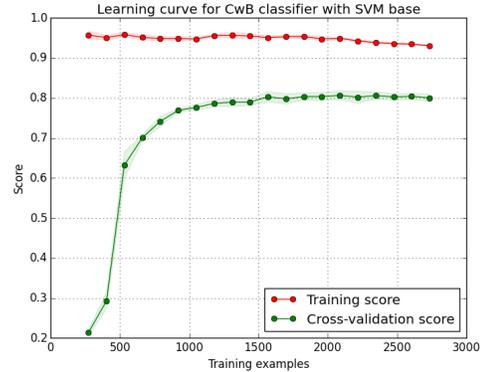


Fig. 7: Learning curve for the CwB classifier with an SVM base and RF experts. Note that the training error increases as the number of training samples increases. The two curves seem to be converging to a value that is far from 1.

This learning curve indicates that the algorithm simply isn't sufficient to classify this data. This can be inferred because the testing error actually increases as the number of training sample increases, and the training and testing error seem to be converging to a value that is far from 1. This was rather surprising, since the LUX classification achieves close to 99% accuracy.

After considering the PCA performed on the data, and the fact that there is a stark difference between the hand-scan and the LUX classification labels, it can be inferred that the labels for the hand-scan data are flawed. While this is good feedback for the LUX collaboration, it meant that the classifier would have to be trained on the LUX classification data, which is the classification that is being tested against.

B. Results based on the LUX classification

The same three off-the-shelf classifiers we re-trained using LUX classification labels. The CwB classifier with an RF base and RF experts, as well as the CwoB classifier, with RF experts were also trained. The accuracy of each classifier, as computed by 4-fold stratified cross-validation, is shown in Table II.

Naive Bayes	0.83 ± 0.01
SVM	0.88 ± 0.01
RF	0.982 ± 0.001
CwB, RF base, RF experts	0.983 ± 0.003
CwoB, RF experts	0.981 ± 0.006

TABLE II: Accuracy of various algorithms in classifying the LUX classification data.

Even with no optimization, the classifiers perform much better on this data. This was somewhat expected because it is easier to split data into only five classes instead of

ten, but it also indicates that the population of each class is more separated in parameter space than it was in the hand-scan data. The CwoB classifier with RF experts was selected for further investigation because it performed with a high accuracy, allowed great freedom over hyper-parameters, and left room for further algorithm enhancement.

The hyper-parameters of each expert classifier were optimized using the Optunity package. Feature selection was not performed on the expert classifiers since the RF method is relatively insensitive to non-predictive features. After optimization, the classifier achieved an accuracy of 0.985 ± 0.004 . A sample confusion matrix is shown below.

$$\begin{bmatrix} 50 & 0 & 0 & 0 & 2 \\ 0 & 119 & 0 & 0 & 1 \\ 0 & 0 & 210 & 0 & 3 \\ 0 & 0 & 0 & 321 & 1 \\ 0 & 0 & 1 & 0 & 23 \end{bmatrix}$$

Fig. 8: Confusion matrix for the CwoB classifier with RF experts. Created by training the classifier on 80% of the training set, and testing on the remaining 20%.

Now, the experts are taking care of all major problem areas that were present before. Still, an occasional signal gets mistaken for noise. It is entirely possible that these pulses were actually mis-classified by the LUX classification, and that this classifier is labeling them correctly. This requires further investigation by the LUX collaboration.

Another metric of classifier success is the precision and recall for each class, which can be seen in Table III.

Class	Precision	Recall
1	1.00	.962
2	1.00	.992
3	.995	.986
4	1.00	.997
5	.767	.958

TABLE III: Precision and recall values for each class, using the CwoB classifier with RF experts.

The precision and recall values only serve as further proof that the classifier is performing well. The weakest point is the precision of class 5. This means that pulses that aren't noise are being classified as noise, or that the experts are missing things. If it is found that these pulses are not mis-labeled, this could be addressed by penalizing the incorrect classification of signal as noise more heavily than the opposite.

The learning curve, shown in Fig. 9, shows an increasing trend in the cross-validation prediction rate as the number of training samples increases. This indicates that the classifier could benefit from more training samples. The low rate of generalization error also indicates that the classifier is likely not over-fitting the data. This is not surprising, since the RF method is not prone to over-fitting.

Though the CwoB classifier does not exceed the LUX classification in accuracy, it does provide some useful information that the current classification process does not. First,

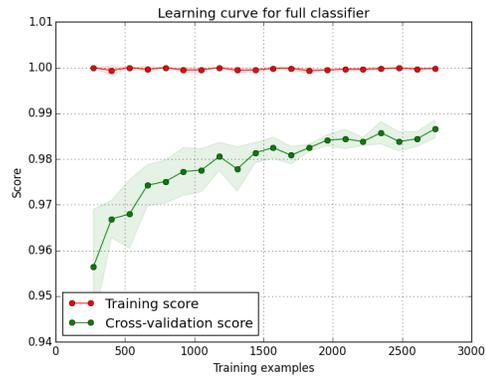


Fig. 9: Learning curve for the CwoB classifier with RF experts. The small difference between the training and CV error shows that the classifier is not over-fitting the data. Additionally, the increasing trend of the CV prediction rate indicates that the classifier would benefit from additional data.

it gives insight into which features are actually important in classification by calculating feature importances. This can aid in understanding what differentiates the physics processes that create these signals. Additionally, the classifier can also give probabilities for a pulse belonging to each class. This would prove useful in the overall data-processing software, especially in the event-building step.

VI. CONCLUSION

One notable insight provide by this study is that humans are bad at doing this classification by hand. If hand-scan data is going to be used to train a classifier or expand the current classifications, it should be labeled only by highly trained scientists and its correctness would need to be verified.

The CwoB classifier with RF experts performs very well, with an accuracy of 0.985 ± 0.004 , and could very likely be improved with the addition of more training samples. This matches the 99% level of accuracy that the current LUX classification achieves, but also gives new, additional information, including the most important features in each classifier and a prediction probability for each class and each pulse, which can be used in the larger analysis framework.

There are many directions that future work could take. The current classifier could be improved or expanded by considering additional ensemble methods, such as Extremely Randomized Trees, AdaBoost, or Gradient Tree Boosting. The rules for the combined classifier could be adjusted to account for the various probabilities assigned by each classifier, or the individual decision trees could be changed to consider a different criterion in splitting nodes. Taking a completely new route, the performance of a neural network could be investigated, or learning could be performed on the time series of the pulse waveform itself.

ACKNOWLEDGMENTS

Many thanks to James Morad for providing the hand-scan data set, and to Mitchell McIntire for his helpful insights.

REFERENCES

- [1] D. S. Akerib, et al., "The Large Underground Xenon (LUX) Experiment," Nuclear Inst. and Methods in Physics Research, A704, pp. 111 - 126, 2013.
- [2] M. Woods, "A Comprehensive Study of the Large Underground Xenon Detector," Ph.D. Dissertation, University of California - Davis, 2014.
- [3] M. Attisha, "Cryogenic Dark Matter Search (CDMS II) - Application of Neural Networks and Wavelets to Event Analysis," Ph.D. Dissertation, Brown University, 2006.
- [4] J. Morad, "pLUX", <http://plux.physics.ucdavis.edu/>
- [5] Pedregosa, et al., "Scikit-learn: Machine Learning in Python," JMLR, 12, pp. 2825-2830, 2011.
- [6] L. Breiman, Random Forests, Machine Learning, 45(1), 5-32, 2001.
- [7] M. Claesen, et al., <http://optunity.readthedocs.org>