Accuracy Correlation in Neutron Resonance Reclassification

Ian Snider[1] Gustavo Nobre[2]
[1]Department of Physics, Truman State University, Kirksville, MO 63501
[2]National Nuclear Data Center, Brookhaven National Lab, Upton, NY 11973

Abstract

The acquisition of accurate neutron resonances is essential for application in practical nuclear systems and understanding astrophysical processes. From these neutron resonances we can determine a description of the neutron interaction cross sections for related nuclei. However, current methods for determining the resonance quantum numbers associated with angular momenta and spin are difficult, time consuming, and may not be fully reproducible, often leading to incorrect spin assignments. To solve this problem, we have employed a machine learning (ML) based method to train an algorithm for identifying and reclassifying incorrect neutron spin assignments. Currently, the algorithm operates with varied successes depending on the isotope, set of features, and training set. For this project, we are examining the properties of the algorithm on polarized Indium-115, in this way, we can accept the given resonance assignments as accurate. We build synthetic data that mimics the statistical properties of real resonances to train the algorithm. Once we have obtained a training accuracy for the synthetic data, we validate the trained algorithm with a set of real In-115 data and observe the correlation between the two sets. However, for unpolarized data, we cannot guarantee the given resonances as accurate, so we also test the trained algorithm on an In-115 set with purposely misclassified resonance assignments. We can then attempt to improve the validation accuracy by adjusting the ML classifier's hyperparameters. We also explored an iterative method in which, under certain conditions, successive reclassifications could incrementally improve the quality of any misclassified resonance sequence.

INTRODUCTION

Cross Sections & Neutron Resonances

A cross section is essentially the effective area of a target as seen by a projectile. As one might expect, the probability of an interaction will increase as the cross section area increases and decrease as the cross section area decreases. The actual area of a cross section depends on the incident energy of a particle and the type of reaction. In the case of this project, the target is an In-115 nuclei and the projectile is a neutron. A neutron reaction occurs when the target nucleus absorbs a neutron and forms a compound nucleus. The compound nucleus then decays, often by emitting a gamma ray (capture channel) or neutron (elastic channel); there are other types of channels as well, such as an inelastic reaction, but resonances from these channels only occur in some cases. In certain energy regions, neutron reactions will lead to a cross section resonance. Specifically, if the incident energy from a neutron reaction approaches an excitation state of the nucleus, then we observe a neutron resonance. On a graph of cross section vs incident neutron energy, this neutron resonance is observed as a "resonance peak".

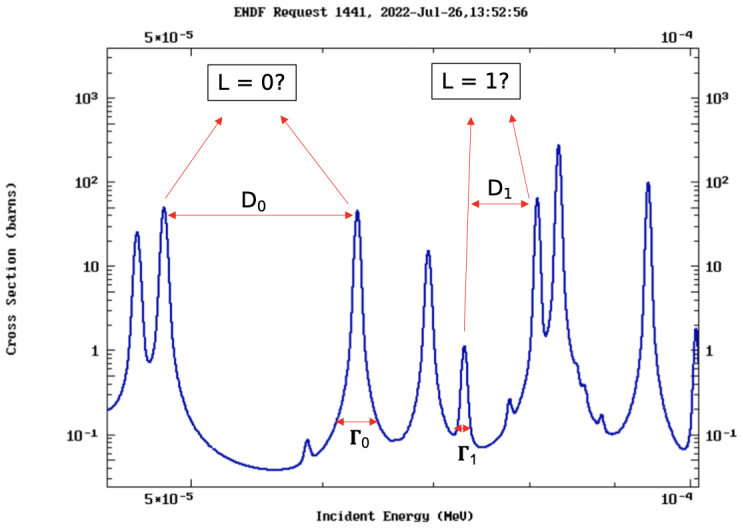


Figure 1: Snippet of nuclear cross section plot for In-115 neutron capture. D and Γ are the resonance spacing and mean width, respectively. Source: Multi- platform EXFOR-CINDA-ENDF Credit: V. Zerkin, IAE-NDS,1999-2022

Nuclear Data & ENDF

In the lab, scientists collect nuclear data by performing experiments that induce these neutron reactions. Once the data is collected, scientists can measure the cross section resonances and estimate the associated quantum numbers for angular momenta and spin through shape analysis. The resonance peaks, as shown in fig. 1, represent different types of waves: s-waves, p-waves, etc, corresponding to angular momenta values of $L = 0$, 1, etc, respectively. These resonances are then compiled into an Evaluated Nuclear Data File (ENDF). The National Nuclear Data Center (NNDC) at Brookhaven National Laboratory (BNL), manages an ENDF repository and organizes the data such that it can be used in many nuclear applications by physicists and engineers. However, there are several issues present in the collection of nuclear data. Fitting the resonances peaks is a manual process that is difficult and can take months or even years. The nature of the collection process lends itself to numerous sources of error, making much of the collected data potentially inaccurate and irreproducible [3]. The *Atlas of Neutron Resonances* [2] is a compilation of evaluated nuclear data written by S.F. Mughabghab at the NNDC. The atlas is filled with gaps in data, typos, and resonances that were simply misclassified. Because the atlas is a valuable resource for research at the NNDC, we saw this as an opportunity to develop a new approach.

Machine Learning & BRR

Machine learning (ML) is a computing process that attempts to learn patterns from a data set in order to make predictions. At the NNDC, we devised a machine learning method to identify and reclassify missassigned neutron resonances: the Bayesian Resonance Reclassifier (BRR) [1]. BRR uses a Python ML library called "Scikit-learn". Scikit-learn is a collection of ML classifiers that offer various methods for learning patterns [4]. For this project, we are interested in the Neural Net and Random Forest classifiers. These classifiers have functions that can be altered using "hyperparameters," values that can be adjusted to optimize the effectiveness of training an algorithm on our data. BRR works by training

the algorithm on synthetic data that mimics the statistical properties of real data. In the code, we can do this by giving the ML classifier a set of features and hyperparameters that we believe will train the algorithm most effectively. Features include items such as capture widths and prior L and J values. This process can be run on a personal computer using a program called: "decide-sequence," or, for large data sets and testing multiple combinations of hyperparameters, on the NNDC cluster by automating decide-sequence [3].

METHODS

ML Training

To train the algorithm, we need resonances for which we know the correct assignments. So, we train BRR on synthetic data based on the statistical properties of In-115. This includes information on the spingroups, max and min energy, resonance spacings, type of projectile (neutron), and the target number of protons and neutrons. For In-115, there are 8 spingroups, which makes this isotope challenging for the ML classifier; other nuclei, such as Cr-52, only have 5 spingroups. But, In-115 does have the advantage of having a maximum L value of 1, whereas Cr-52 has a maximum L value of 2.

However, synthetic data does not contain any missassigned resonances and a perfect data set does not correspond to the problem we are trying to solve. ENDF files have errors, so in order to give the algorithm the best quality of training, we purposely missassign a certain percentage of resonances in a process called "jumbling" [1]. This allows us to know whether or not a resonance was reclassified by the algorithm correctly.

Polarized In-115

Ultimately, the goal is to know the correct spin assignments of the resonances, if we could always determine the correct spin assignment when evaluating the cross section then there

would be no need for this project. There is a method of doing this, but it has specific uses and is very expensive. If you polarize a compound nucleus in a polarized-beam experiment, then it is possible to know the spin assignment of a resonance prior to analyzing the cross section, however the process for doing this involves expensive equipment, and is not usually worth the cost. But, we do have some polarized In-115 data at our disposal. Because it is polarized, we can accept the given spin assignments as accurate. This allows us to validate the trained algorithm on real experimental data with accurate resonance assignments.

Validating Training Accuracy

We are most interested in observing how a validation set performs on a trained algorithm. To do this, we train an algorithm on synthetic data at multiple values of jumbleness. We then test the trained algorithm at each jumbleness value on the validation set, which, in this case, is a set of unjumbled polarized In-115 data. In this test, we will compare hyperparameter combinations for the Neural Net and Random Forest classifiers. Once we have obtained the training and validation accuracy, we will examine the correlation between the two sets and observe how the correlation changes as a function of the synthetic training jumbleness. There are many hyperparameters for each classifier that we could test, but ultimately we are just interested in how the training accuracy correlates to the validation accuracy. Ideally, we will have a validation accuracy that is consistently correlated with the training accuracy at each training jumbleness. Additionally, we would like to reproduce results from a previous study on BRR, in which the validation accuracy of an algorithm trained using the Random Forest classifier was tested.

Jumbling the Validation Data

Generally, polarized data will not be available to validate the training accuracy. To account for this, we jumble the validation data and perform a similar test as the one above.

We can then observe a validation accuracy as a function of the training and validation jumbleness. For this test we will use the Random Forest classifier. At this point, the classifier we choose is less important, we are solely interested in the relationship between the training and validation accuracy. The Random Forest classifier tends to be a little more sporadic than Neural Net when generating a training accuracy. This is useful because it allows us to see how averaging multiple reclassification cycles affects the smoothness of the overall validation accuracy. Once we have obtained validation accuracies for jumbled validation data, we can examine the results for potential paths that will optimize the convergence accuracy of iterative reclassification.

Iterative Reclassification

Now that we have achieved some amount of success with the ML method, we would like to see how we can expand its capabilities. One idea is an iterative reclassification method, in which we perform successive reclassifications that, under certain conditions, could incrementally improve the quality of any misclassified resonance sequence. Essentially, we take a reclassified validation data set and reclassify it again, forming a new validation set, which we reclassify again, forming another data set, and we repeat this process for a specified number of iterations. It is also important to note that we are only performing successive reclassifications on the validation data, the training data will only be reclassified from its initial data set at each iteration. If we start with a set of validation data initially jumbled at 80%, ideally we can incrementally improve the accuracy from 20% to a value higher than previously possible with a single iteration. This method could also allow us to see which specific resonances the algorithm favors for reclassification, which would be tremendously insightful for the development of BRR. Currently, we only have the capability of testing one training jumbleness value at a time, and we can only use one reclassification cycle per iteration since we currently do not have a method of creating a generalized data set from the average resonance assignments of multiple cycles. So, we are limited to iteratively

reclassifying a single reclassification cycle.

RESULTS

Validating Training Accuracy

Initially, I tested the capabilities of the Neural Net classifier on polarized In-115 validation data because Neural Net tended to have to best validation score. Over the course of the project however, Random Forest proved more interesting because it was less accurate and therefore made optimizations easier to find. But the first set of data comes from a Neural Net test of the regularization and solver hyperparameters. Regularization is a hyperparameter that helps the algorithm avoid overfitting for weights with large magnitudes. Essentially, if I have a small regularization value, it will perform well on a single data set, but will not perform well when transferring to another, so the training accuracy will be good, but the validation accuracy will not. This is why it is best to have a mid-ranged value for the regularization parameter "alpha." The solver also performs a similar task that attempts to perform weight optimization. Testing the two together can reveal potentially useful points for improving the overall effectiveness of BRR.

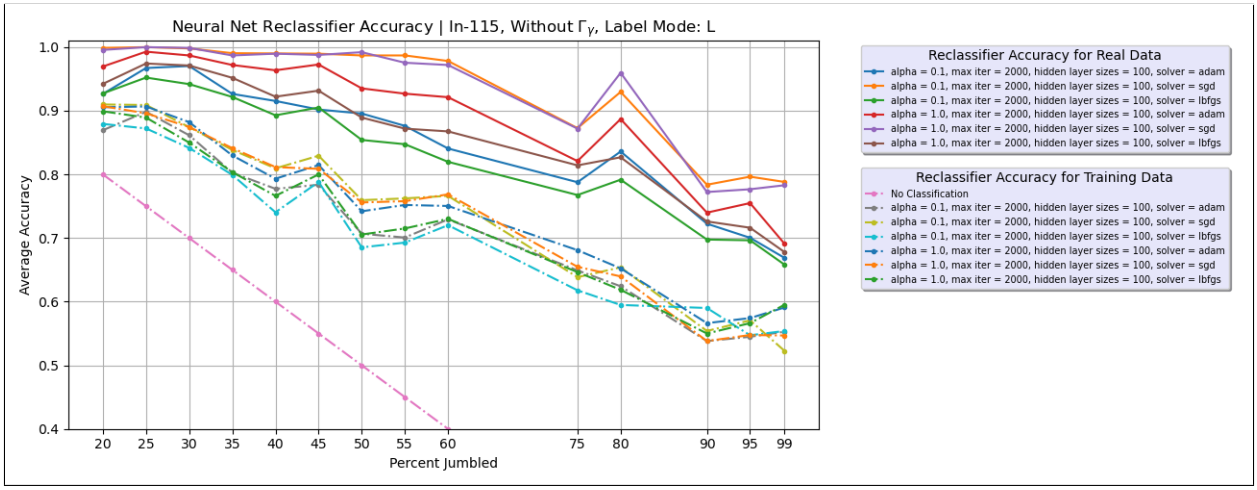


Figure 2: n = 1 cycle. A comparison between the validation and training accuracy of the Neural Net classifier. Generally, sgd solver was most effective, with little difference in alpha.

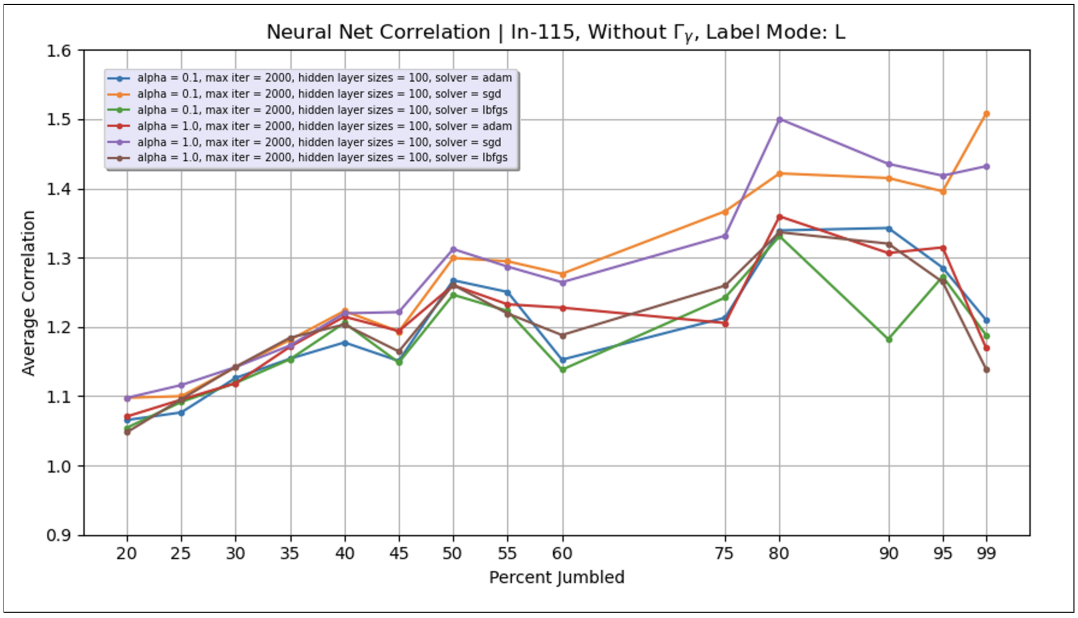


Figure 3: n = 1 cycle. Correlation is calculated as $correlation = \frac{validation}{training}$. The hyperparameter correlation diverges at high training jumbleness values

For these two plots we see that correlation between the validation and training accuracies degrades as the training jumbleness increases. For the next two plots, we use 10 cycles to improve the average.

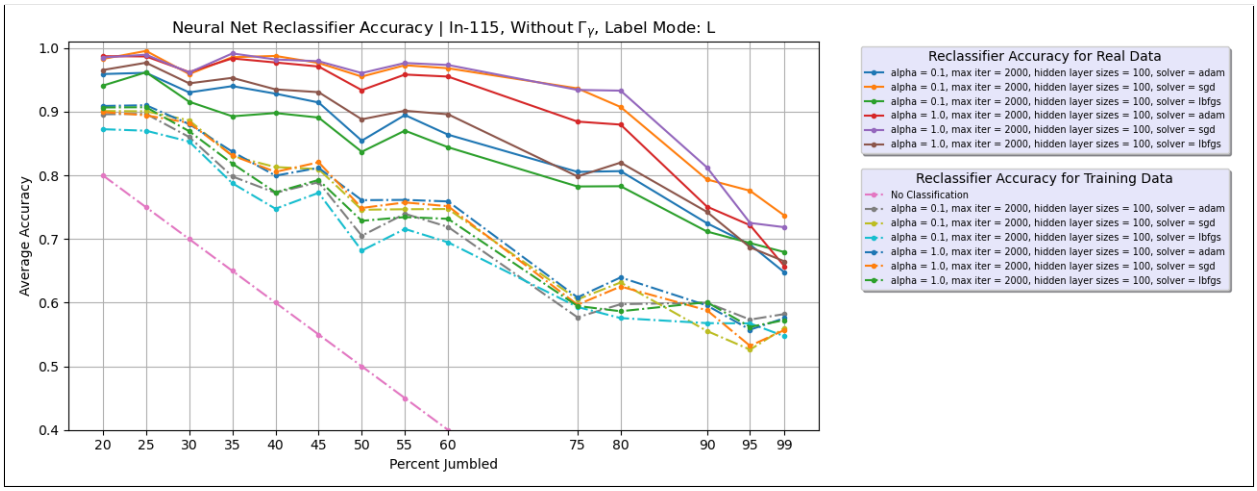


Figure 4: n = 10 cycles. An average of 10 cycles produces a smoother result for the plot.

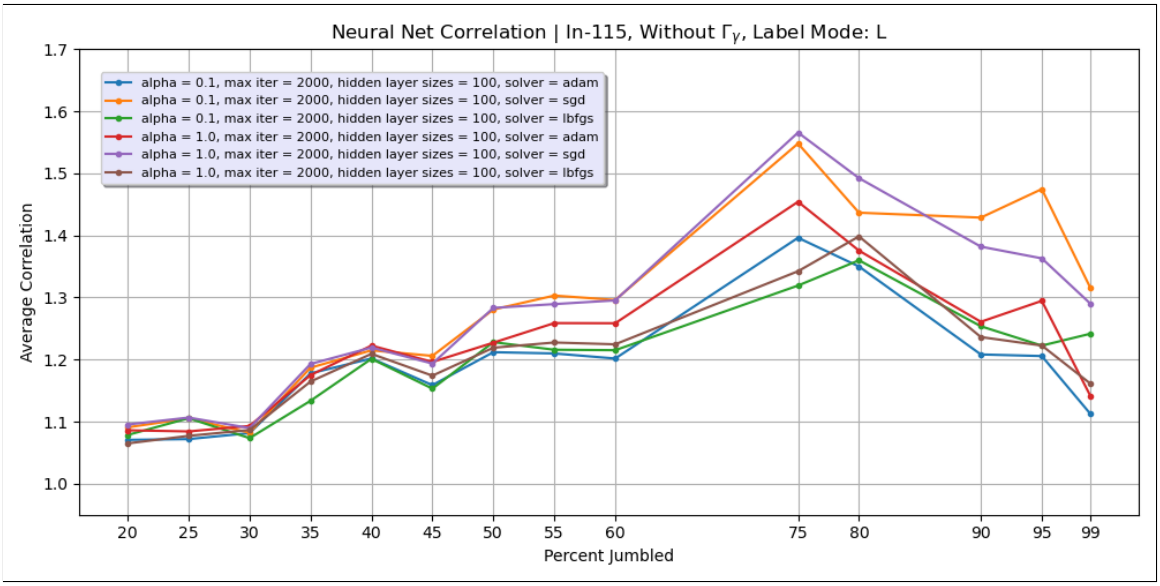


Figure 5: n = 10 cycles. Correlation is calculated as $correlation = \frac{validation}{training}$. We see less divergence in the hyperparameters when we average more cycles.

Switching to the Random Forest classifier will result in an overall decrease in the average accuracy, but the validation score may correlate more closely with the training accuracy. The following accuracies are the result of averaging 500 cycles.

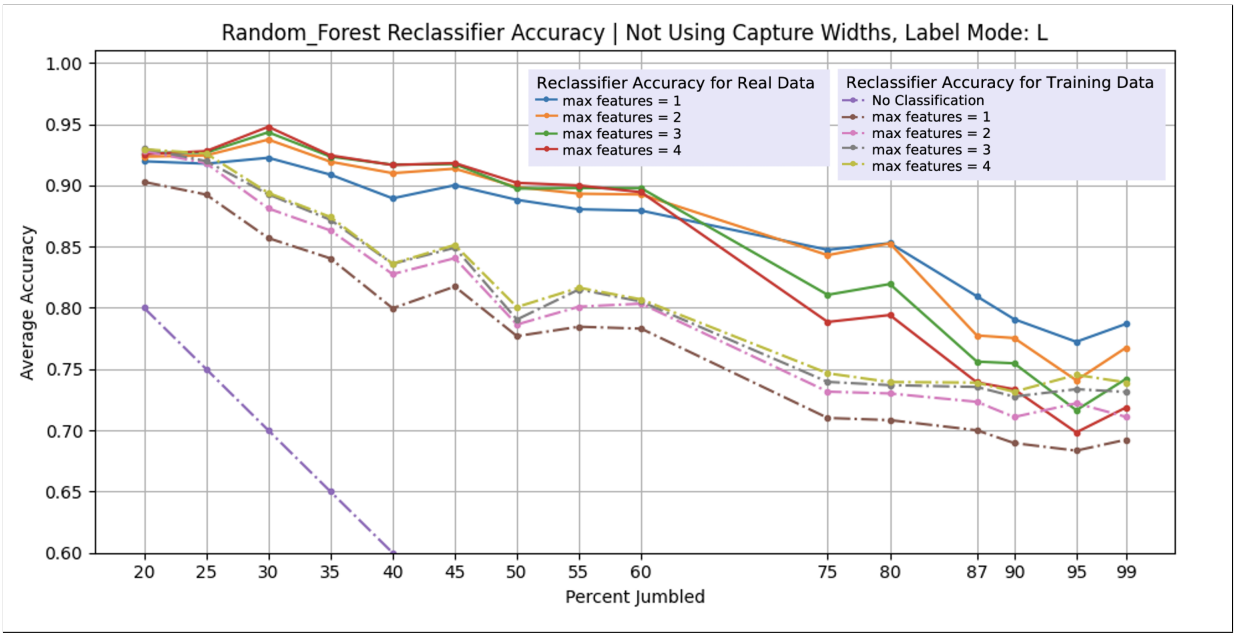


Figure 6: n = 500 cycles. Max features tells the Random Forest how many features to use when looking for a place to split a branch. Interestingly, max features greater than 2 tended to hurt the validation score.

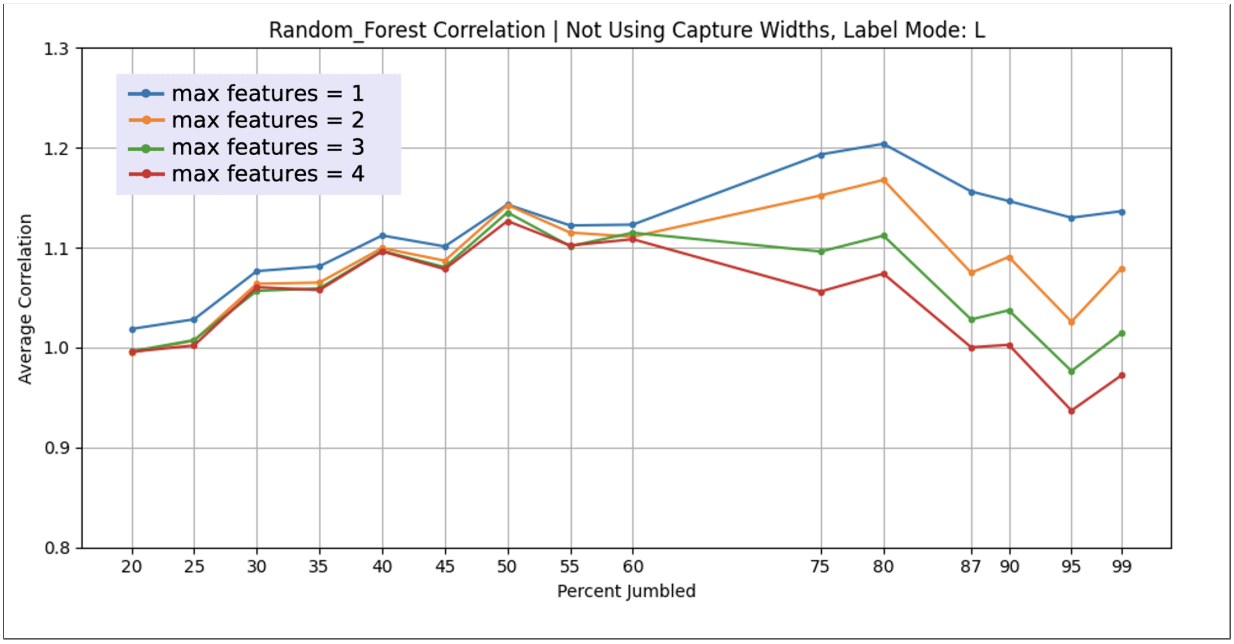


Figure 7: n = 500 cycles. This correlation plot produces a result similar to Neural Net. We again see the correlation degrade at high jumbleness values.

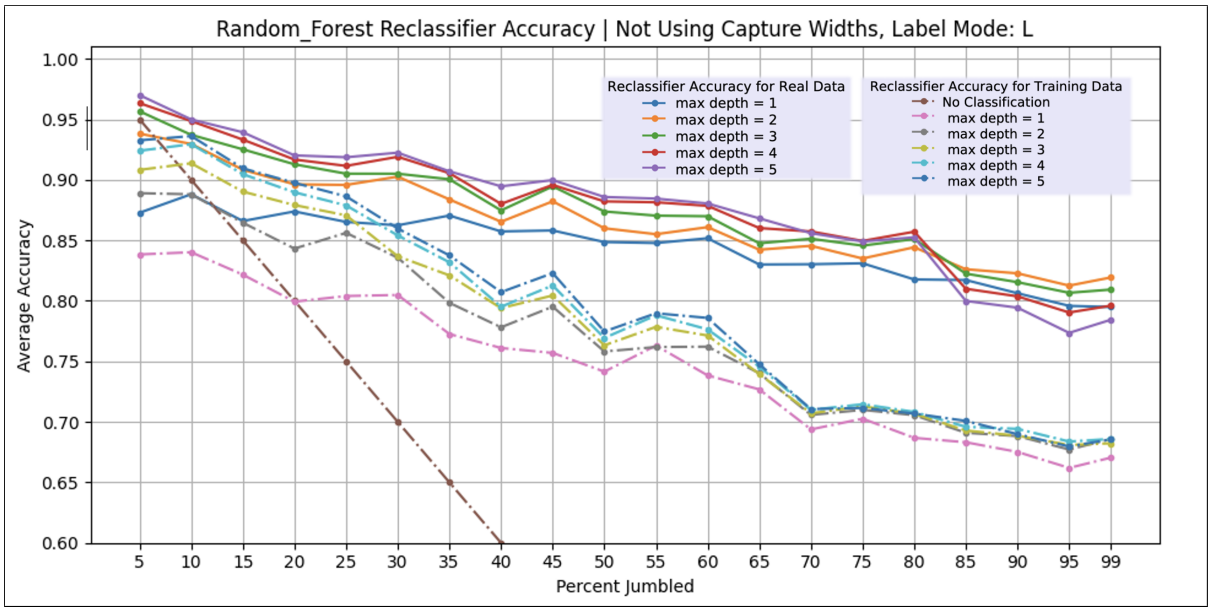


Figure 8: n = 500 cycles. Max depth tells the Random Forest classifier the maximum number of levels that are allow to be split into new nodes. This figure shows a widening gap between the training and validation accuracy.

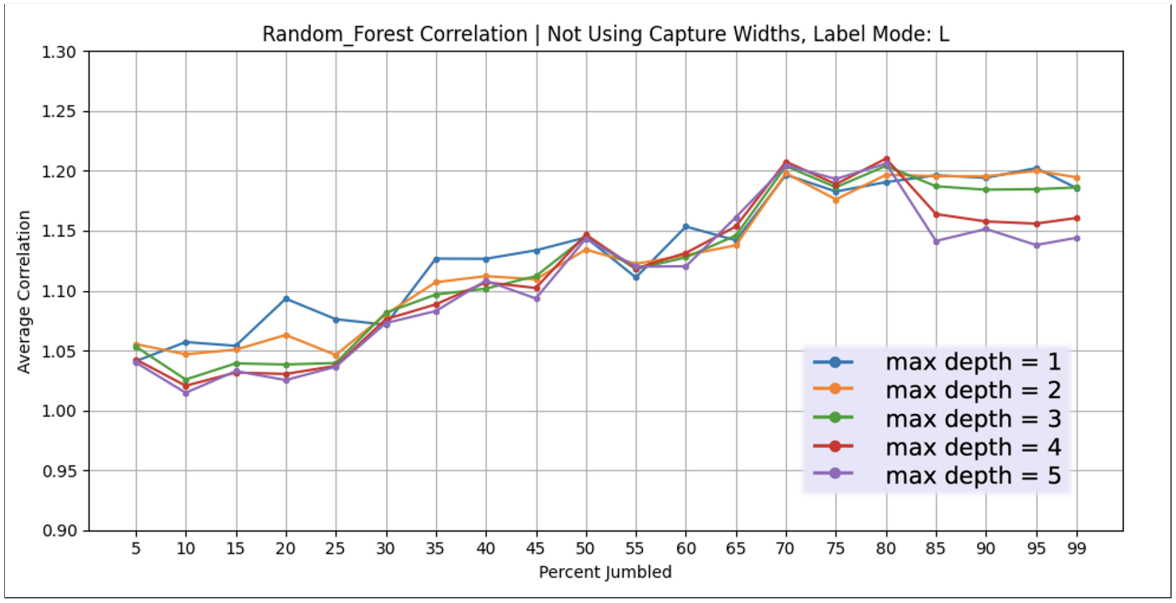


Figure 9: n = 500 cycles. Here the correlation plot degrades at higher training jumbleness values. The training accuracy falls off faster than the validation accuracy.

Interestingly, just about every case shows the validation data performing better than the training data. While this is good for accuracy, it is not necessarily ideal for predictability. A stronger correlation between the training and validation sets would indicate that the training set is a good representation of reality. For low jumbleness values, the synthetic data seems to be a near perfect representation of reality, but as the jumbleness values increase, the correlation almost always degrades, consistently returning a higher validation accuracy than training accuracy.

Jumbling the Validation Data

For the jumbling the validation data, I took a set of polarized In-115 real data and used the jumbling process to produce a jumbled set of data. I repeated this process with multiple jumbleness values in order to reveal trends in the validation accuracy. Additionally, I also tested using jumbled synthetic data as the validation set in hopes of revealing similarities between a set of real and synthetic data. For simplicity, I first plotted a few jumbleness values and their accuracies for preliminary observations.

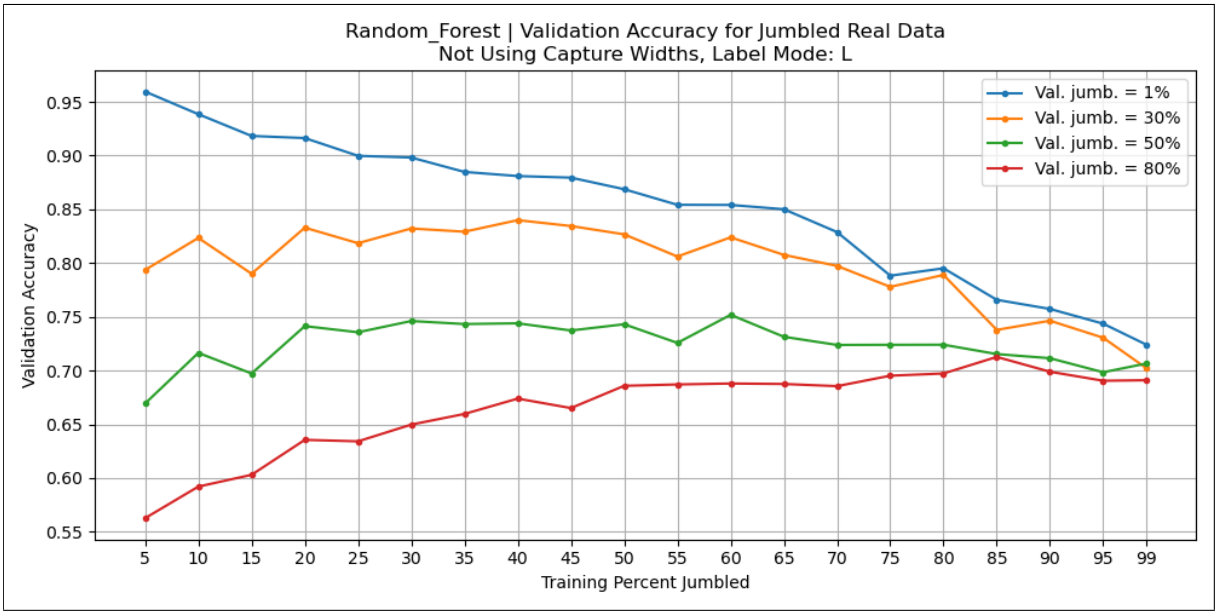
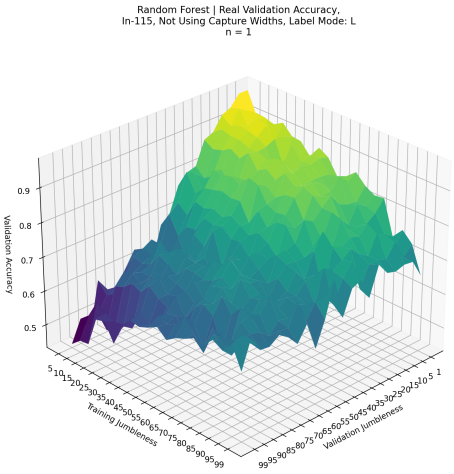
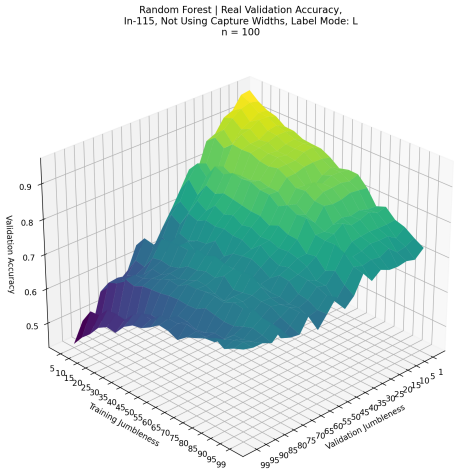


Figure 10: n = 100 cycles. This plot demonstrates how different values of jumbled real data converge at the same point of training jumbleness.

We can now expand to include more validation jumbleness values and plot validation accuracy as a function of validation and training jumbleness on a 3D graph. Additionally, we can compare the effects of using a few vs many cycles for averaging the validation accuracy.

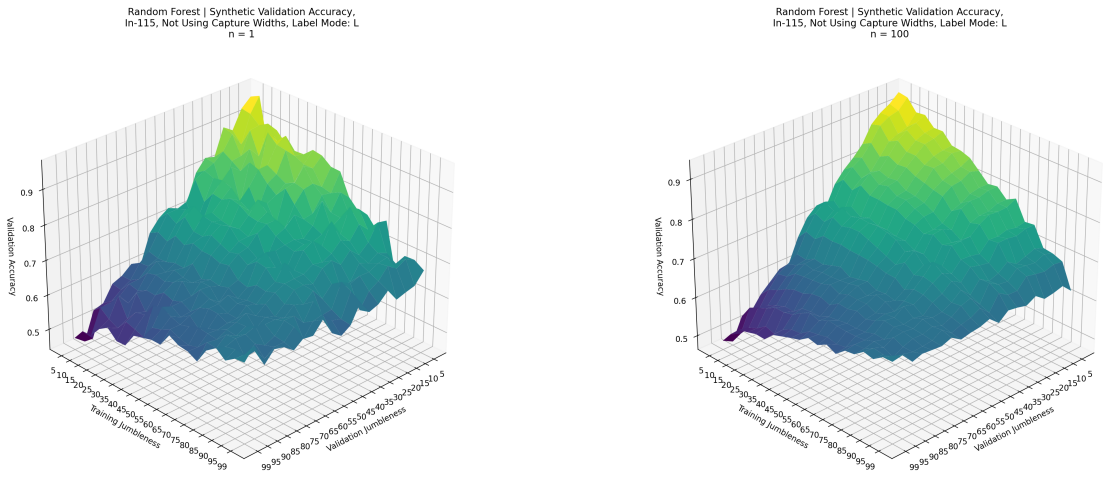


(a) n = 1



(b) n = 100

Figure 11: Jumbled real In-115 validation data



(a) n = 1 (b) n = 100

Figure 12: Jumbled synthetic In-115 validation data

The validation accuracies calculated for these plots were collected using the Random Forest classifier. Random Forest has sporadic behavior which is evident in the n = 1 plots, which is why it is important to take an average when reclassifying validation data. From the real validation plot with 100 cycles, we can see that a validation jumbleness around 50% will perform about the same for any training jumbleness, which is suggested by the convergence patterns seen in fig. 10. It is also interesting to note the similarities when validating with synthetic data. We see that the plots have the same shape, but the synthetic data is smoother. This could be because it has more resonances than the real In-115 data set and because the synthetic validation data has the same statistical properties as the training data.

Iterative Reclassification

For the iterative reclassification tests, we start with an original validation jumbleness value of 80% and test various values of training jumbleness. There are also additional figures and information about iterative reclassification in the appendix. Some of this information

will pertain more towards future work.

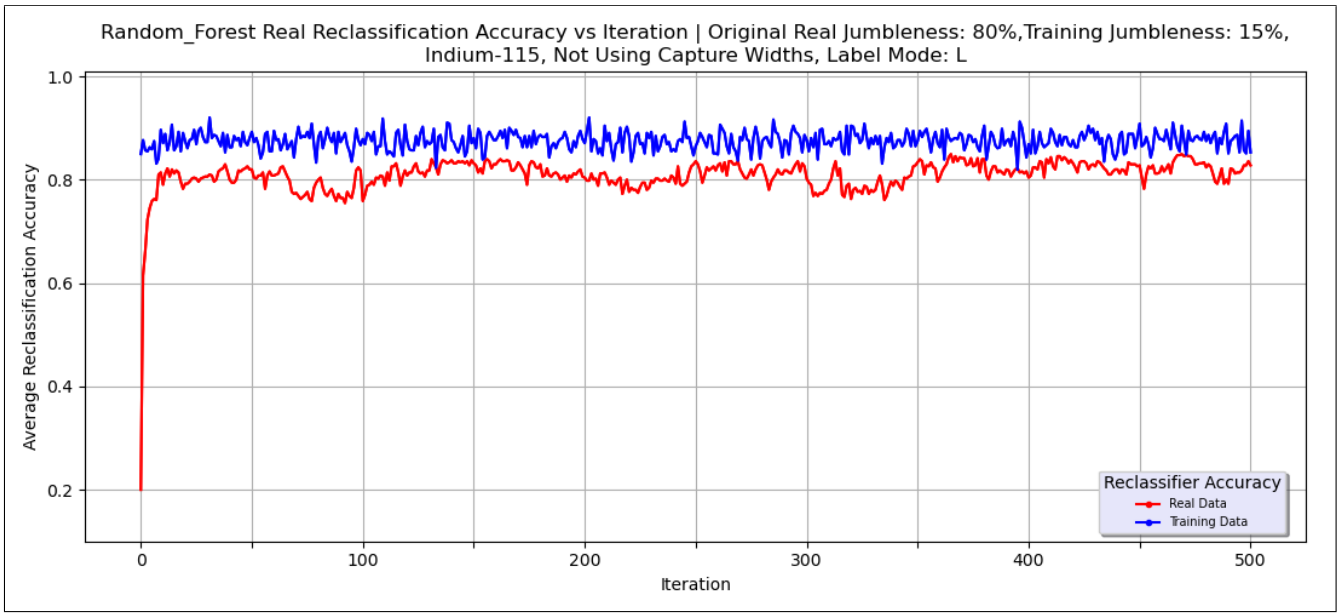


Figure 13: 500 iterations. After only 25 iterations we see that the validation accuracy begins to converge at around 80% accuracy.

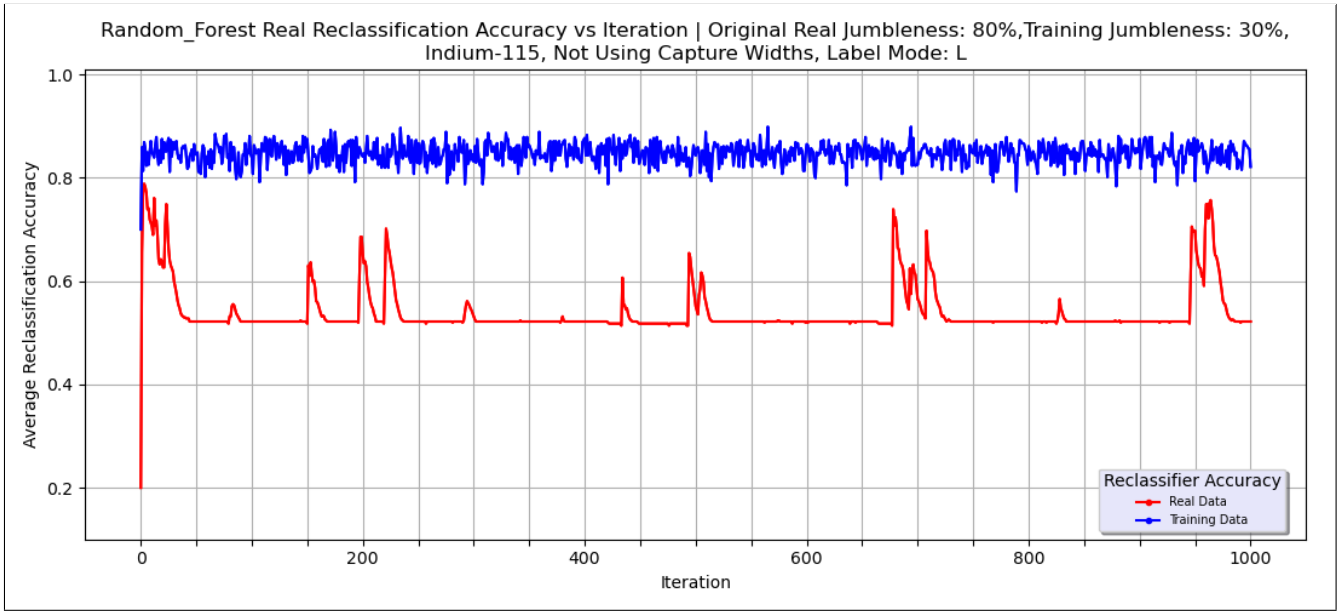


Figure 14: 1000 iterations. The validation accuracy begins to converge around 53% accuracy at 50 iterations. Additionally, we observe sudden spikes in the validation accuracy.

In fig. 13, we observe an expected behavior for successive reclassifications, the validation accuracy improves steadily for a few iterations then converges on a value below the training

accuracy. But, in fig. 14, we observe strange behavior. The accuracy in the reclassified validation data improves drastically in the first few iterations but a sharp decline in accuracy quickly follows. In the appendix, there are more graphs showing that this behavior begins as a function of the training jumbleness. As the training jumbleness increases, this behavior becomes more likely. There is also the issue of the sudden spikes in accuracy that occur at seemingly random intervals throughout the successive reclassifications. Currently, we can only speculate as to the cause.

Discussion & Conclusion

When correlating the validation and training accuracies, the validation accuracy was consistently higher than the training accuracy. Because the validation data does not perfectly correlate to the training set, this makes the validation accuracy hard to predict. There are several possible causes for this, but one likely cause may be the size discrepancies between the validation and training set. The given polarized In-115 data set only contains around 250 resonances while the training set contains nearly 700. This means that the algorithm may be overfitting the training set and thus is not perfectly applicable to the validation set. If the validation and training sets had a similar amount of resonances, the correlation may improve.

Jumbling the validation data provided insight on testing the relcassifier algorithm for varying degrees of evaluated nuclear data. We were able to show that the algorithm was somewhat able to accurately reclassify data with many missassigned resonances. This is useful because it means we are on track to being able to reclassify any set of data, regardless of how inaccurate the initial evaluation was. These results are promising and we will continue to examine and develop BRR's performance with reclassifying jumbled data.

Successive iterations has proved to be a valuable and interest part of this investigation. The accuracy spikes occur when the algorithm reclassifies L from 0 to 1. We currently, do not fully understand why this is happening, but we suspect it is because of the averaging.

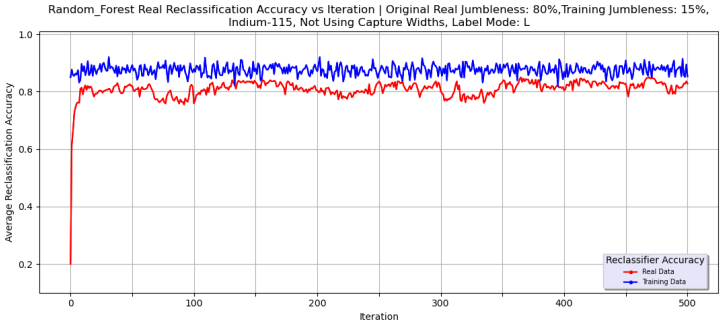
As mentioned in the methods, the successive reclassifications only takes data from a single reclassified file. When testing accuracy correlation we were able to take an average of many cycles to produce smooth data, but here we cannot do that. Instead, if we wanted to run multiple cycles, we would need to compute the most common resonance sequence for each resonance in the reclassified file. Then, from this calculation, we could make a new data set representing the most probable outcomes for each resonance sequence. If we could do this, then perhaps the accuracy spikes would disappear and the validation accuracy would converge at an overall higher value. However, we have not yet implemented this so it currently resides in the realm of future work. We also briefly examined the properties of individual reclassified resonances. We noticed a strong correlation between neutron widths and the assignment status of each resonance. In the appendix, we have created heat maps of the assignment status as a function of neutron width and current iteration. This is also something we would like to investigate further as understanding the individual reclassified resonance could reveal possible improvements in our training sets and ML process.

Acknowledgements

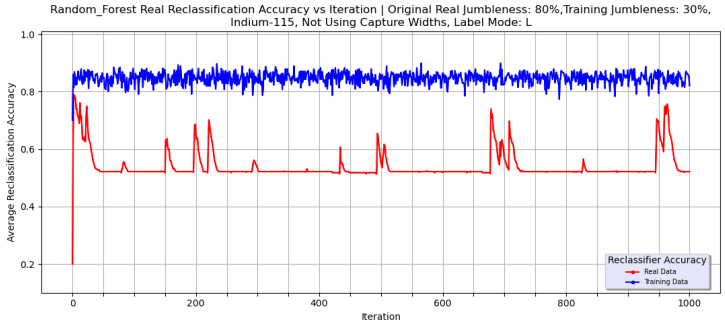
I would like to sincerely thank my mentor, Dr. Gustavo Nobre, for his insight and guidance throughout the summer SULI program. I would also like to thank my fellow intern, Cole Fritsch, all past interns, and Dr. David Brown for their combined efforts and contributions to this project. This project was supported in part by the U.S. Department of Energy, Office of Science, Office of Workforce Development for Teachers and Scientists (WDTS) under the Science Undergraduate Laboratory Internships Program (SULI).

Appendix

Iteration Figures with L-Value Heat Maps for Real Validation

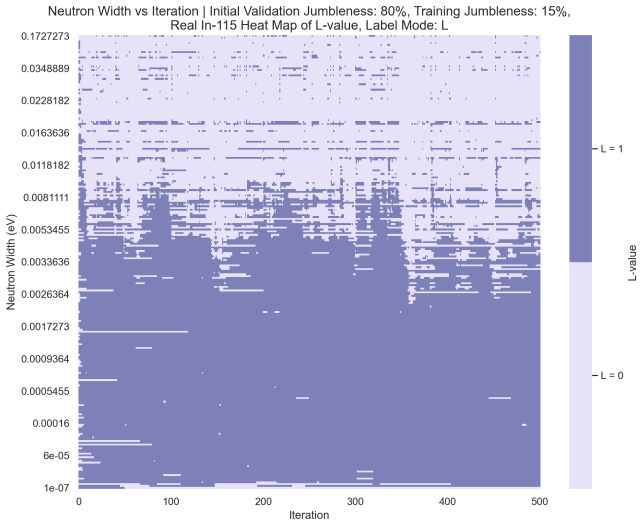


(a) Training Jumbleness = 15%

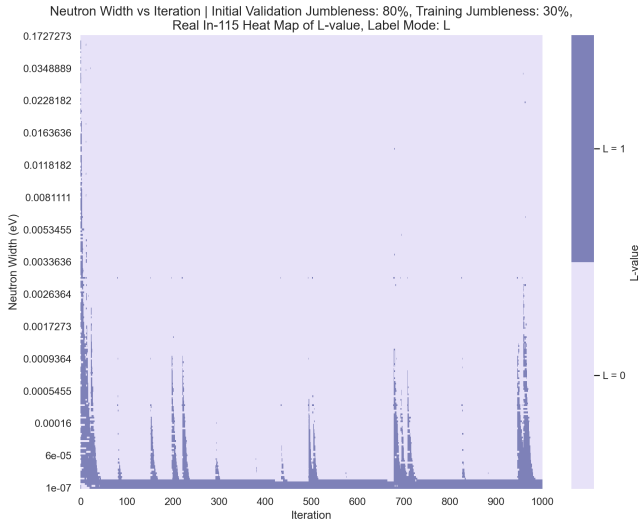


(b) Training Jumbleness = 30%

Figure 15: Iterative reclassification tested on 80% jumbled real validation data



(a) Training Jumbleness = 15%

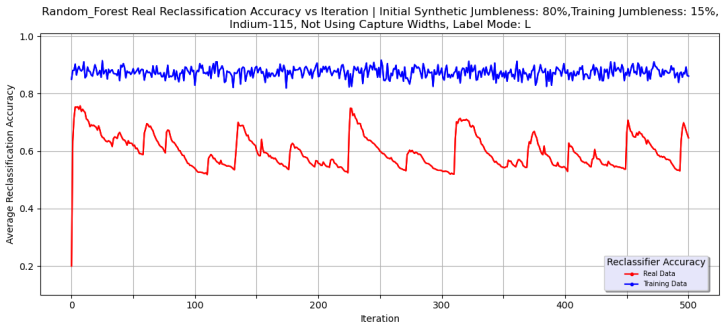


(b) Training Jumbleness = 30%

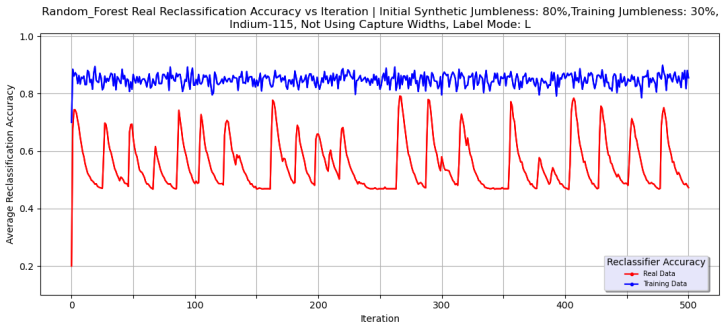
Figure 16: Density of L-Value plotted as a function of neutron width vs iteration. Includes initial values.

For low training jumbleness values, the spread of $L = 0$ and $L = 1$, is mostly even. But, when the training jumbleness is increased, the algorithm begins to almost entirely favor $L = 0$ over $L = 1$ except in the case of low neutron widths.

Iteration Figures with L-Value Heat Maps for Synthetic Validation

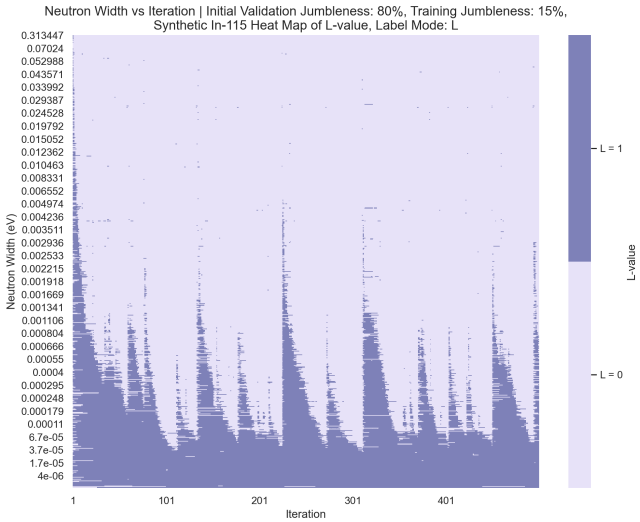


(a) Training jumbleness = 15%

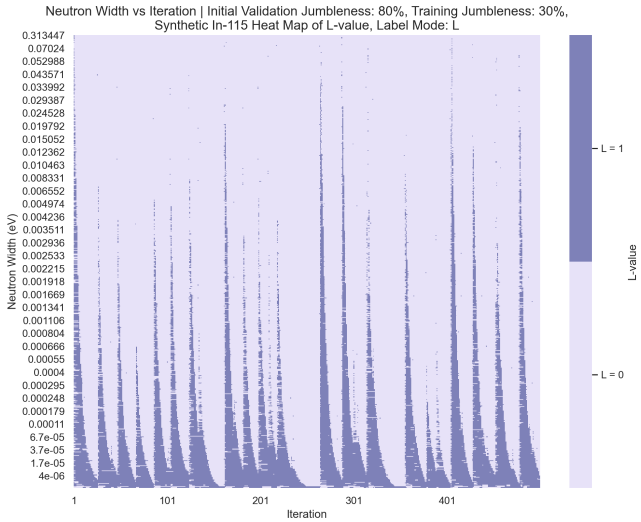


(b) Training jumbleness = 30%

Figure 17: Iterative reclassification tested on 80% jumbled synthetic validation data



(a) Training jumbleness = 15%

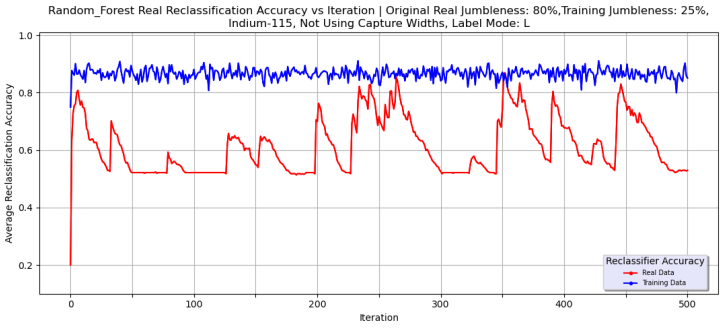


(b) Training jumbleness = 30%

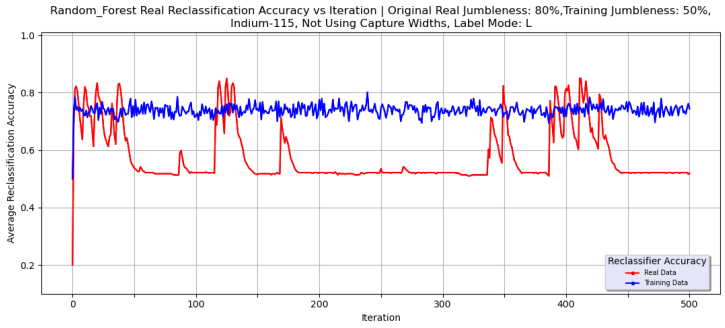
Figure 18: Density of L-Value plotted as a function of neutron width vs iteration. Synthetic heat maps do not contain initial values.

For synthetic validation data, we observe a similar trend in both the iteration plots and heat maps of L. And, we again see that the algorithm favors $L = 0$ over $L = 1$. However, the sharp accuracy peaks begin at a lower value of training jumbleness when compared to real validation data. Additionally, the accuracy peaks are more common and taller.

Additional Iteration Figures with Assignment Status Heat Maps for Real Validation

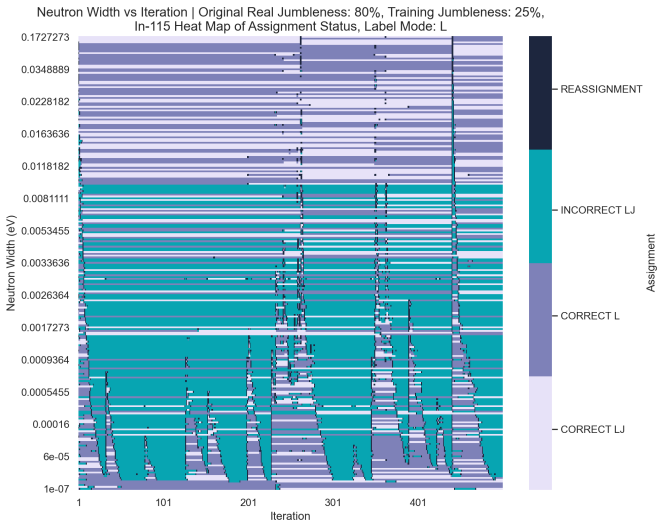


(a) Training jumbleness = 25%

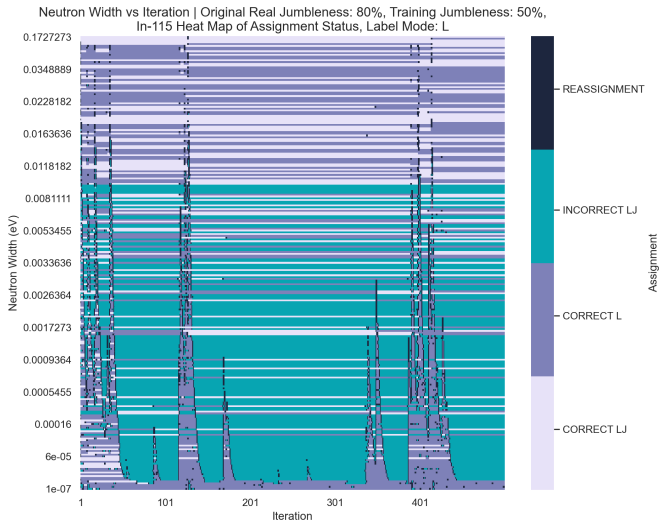


(b) Training jumbleness = 50%

Figure 19: Iterative reclassification tested on 80% jumbled real validation data



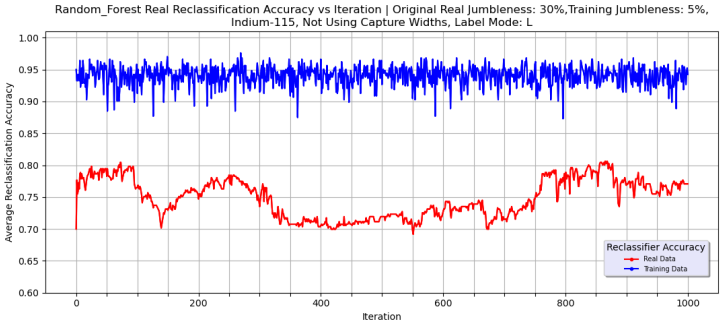
(a) Training jumbleness = 25%



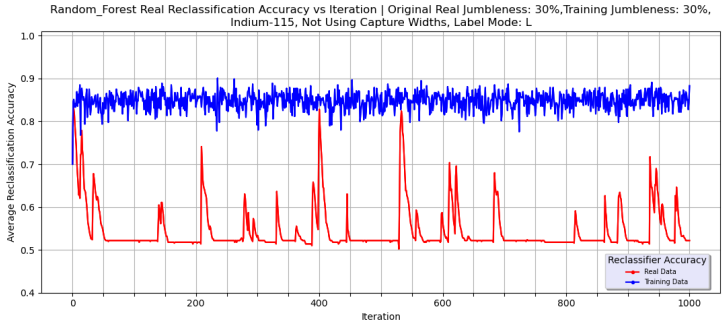
(b) Training jumbleness = 50%

Figure 20: Density of Assignment Status plotted as a function of neutron width vs iteration.

This is an earlier version of the L-Value heat maps. I first tried plotting out the status of each resonance to see which neutron widths were reclassified most often, but ultimately the L-Values were most useful for revealing potential issues with iterative reclassification.



(a) Training jumbleness = 5%



(b) Training jumbleness = 30%

Figure 21: Iterative reclassification tested on 30% jumbled real validation data

References

[1] D. Brown, G. P. A. Nobre, S. J. Hollick, P. Rodriguez, and S. Scoville, "Machine learning applied to classifying neutron resonances." [Online]. Available: https://www.osti.gov/biblio/1673302

[2] S. F. Mughabghab, *Atlas of Neutron Resonances, Vol. 1 and 2.* Elsevier, 2018.

[3] G. NOBRE, G. Nobre, D. Brown, S. Scoville, M. Fucci, S. Ruiz, R. Crawford, A. Coles, and M. Vorabbi, "Expansion of machine-learning method for classifying neutron resonances." [Online]. Available: https://www.osti.gov/biblio/1823638

[4] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.