Spring 5-12-2023

Exploration of Feature Selection Techniques in Machine Learning Models on HPTLC Images for Rule Extraction

Bozidar-Brannan Kovachev

Exploration of Feature Selection Techniques in Machine Learning Models

on HPTLC Images for Rule Extraction

by

Brannan Kovachev

A thesis submitted to the faculty of The University of Mississippi in partial fulfillment of the requirements of the Sally McDonnel Barksdale Honors College

Oxford

May 2023

Approved by

_____

Advisor: Dr. Yixin Chen

_____

Reader: Dr. Feng Wang

_____

Reader: Dr. Thai Le

ABSTRACT

Research related to Biology often utilizes machine learning models that are ultimately uninterpretable by the researcher. It would be helpful if researchers could leverage the same computing power but instead gain specific insight into decision-making to gain a deeper understanding of their domain knowledge. This paper seeks to select features and derive rules from a machine learning classification problem in biochemistry. The specific point of interest is five species of *Glycyrrhiza*, or Licorice, and the ability to classify them using High-Performance Thin Layer Chromatography (HPTLC) images. These images were taken using HPTLC methods under varying conditions to provide eight unique views of each species. Each view contains 24 samples with varying counts of the individual species. There are a few techniques applied for feature selection and rule extraction. The first two are based on methods recently pioneered and presented as "Binary Encoding of Random Forests" and "Rule Extraction using Sparse Encoding" (Liu 2012). In addition, an independently developed technique called "Interval Extraction and Consolidation" was applied, which was conceptualized due to the particular nature of the dataset. Altogether, these techniques used in consort with standard machine learning models could narrow a feature space from around one-thousand candidates to only ten. These ten most critical features were then used to derive a set of rules for the classification of the five species of licorice. Regarding feature selection, compared to standard model parameter optimization, the Binary Encoding of Random Forests performed similarly, if not much better, in reducing the feature space in almost all cases. Additionally, the application of Interval Extraction and Consolidation excelled in further simplifying the reduced feature space, often by another factor of five to ten. The selected features were then used for relatively simple rule extraction using decision trees, allowing for a more interpretable model.

ACKNOWLEDGEMENTS

I would like to thank Dr. Yixin Chen for serving as my thesis advisor and providing invaluable assistance, guidance, and feedback throughout the entire process. I would like to thank Dr. Chittiboyina for providing us with the dataset and a problem to explore. Finally, thank you to all the teachers and mentors who have helped shape the foundation of knowledge this paper was built on.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

SECTION 1

INTRODUCTION

Much biological research sets out to solve and understand a problem. Often, solving the problem with high accuracy can be achieved through machine learning models, such as using a neural network to perform multiclass classification. However, many models like neural networks, are “black boxes” and do not provide interpretable results. In contrast, other models, such as Decision Trees, are easily interpretable but are significantly less accurate. This paper investigates methods for producing highly accurate and easily interpretable results and models.

Regarding multiclass classification, decision tree models allow for simple rule extraction as each node compares a selected feature against a threshold value to classify the sample ultimately. However, they can suffer in accuracy due to their high variance, especially when compared to neural networks or RF models (Talari 2022). This is often due to noisy input data from a large feature space. Therefore, if we can create a compact representation of the feature space that uses only the most valuable features, we can produce highly interpretable models that are unlikely to be overfitting.

The compact representation of our feature space will be achieved through various feature selection methods. Most generally, feature selection is “the process of reducing the number of input variables when developing a predictive model” (Brownlee 2020). This paper primarily uses two varieties of feature selection to narrow down the space. After selecting only the most critical features for classification, we can train decision trees on these features to produce a relatively lower variance and higher accuracy models when compared to using the entire feature set.

Furthermore, these decision trees allow for simple rule extraction when compared to the alternative and also highly accurate, low variance “black box” classifiers such as SVM or RF.

For model interpretability, we seek to maintain the benefits of rule extraction from decision trees while making up for their failings through strict feature selection.

SECTION 2

DATA

This paper investigates the classification of five *Glycyrrhiza* species, more commonly known as licorice. Various samples of these five species were photographed through High-Performance Thin Layer Chromatography (HPTLC) techniques.

2.1 HPTLC Introduction

HPTLC is a way to separate and identify different substances in a mixture. This is done by using a special kind of paper or plate and a solvent to help see the different substances better. The outcome of HPTLC is a separation of the different substances in a mixture, resulting in individual bands on the paper or plate. The position and size of these bands can be used to identify the different substances in the mixture. Typically, this is done by comparing the bands or spots to those of known substances or by using special tools to measure the bands or spots. The information obtained from HPTLC can be used to identify unknown substances, quantify the amount of a particular substance present in a mixture, or study the purity of a substance (Marathe 2020). Furthermore, this method of analysis has seen considerable use and growth in popularity over the past decade, so the development of interpretable models from this data is of utmost interest (Sathyajith 2019). In our application, we will be using the HPTLC images as input into machine learning models to classify a sample into one of five species of licorice.

2.2 *Glycyrrhiza* HPTLC Samples

The dataset we used does contain five species of *Glycyrrhiza* in total however, they are not of equal distribution. The species name and sample count of each can be seen in Table 2.1:

Table 2.1 Sample Count per Species of *Glycyrrhiza*

Name	Count
G. glabra	6
G. inflata	6
G. uralensis	6
G. echinata	3
G. lepidota	3

Altogether, this is a total of 24 samples.

2.3 Varying Conditions of HPTLC Images

Rather than just a single set of 24 samples, we used several HPTLC images under various conditions with the same count of samples for each species per condition. The sample conditions varied in whether they were analyzed for polar or non-polar constituents, the wavelength of light the images were taken under, and whether they were taken prior or post derivatization. Table 2.2 Contains the full set of conditions.

Table 2.2 Conditions under which HPTLC Images were Taken

Light	**Polar vs Non-Polar Properties**	**Prior or Post Derivatization**
White Light	Non-Polar	Post
UV 254nm	Non-Polar	Prior
UV 366nm	Non-Polar	Post
UV 366nm	Non-Polar	Prior
White	Polar	Post
UV 254nm	Polar	Prior
UV 366nm	Polar	Post
UV 366nm	Polar	Prior

Each of these images of 24 samples will be referred to as a "view." For the reader's benefit, we have provided three examples from this table in Figures 2.1 – 2.3.

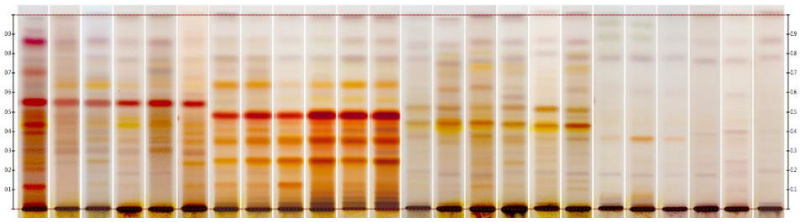

Figure 2.1: HPTLC image under the Conditions of White, Non-Polar, and Post Derivatization

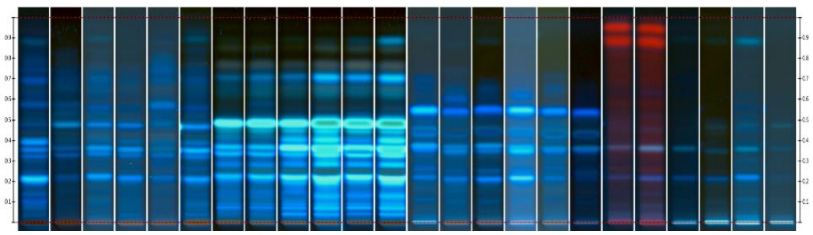

Figure 2.2: HPTLC image under the Conditions of UV 366nm, Non-Polar, Prior Derivatization

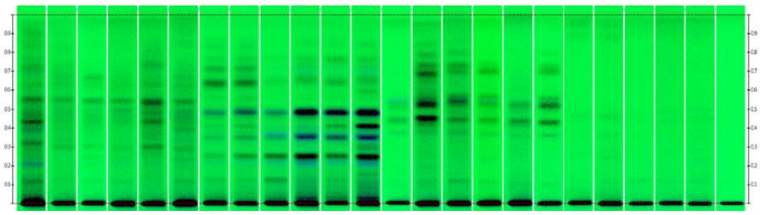

Figure 2.3: HPTLC image under the Conditions of UV 254nm, Non-Polar, Prior

Each vertical segment is a sample, and each of these views contains 24 samples.

SECTION 3

DATA PREPARATION

Prior to applying any feature selection algorithms to the data, we prepared and transformed each sample through a few standard practices.

3.1 Dimensionality Reduction

Though the original images are in color, we seek to reduce the feature space, so the first transformation we applied was a change from color to greyscale. Rather than using three values per pixel, we will use only one. Not only does this limit the feature space, but it also helps create a more standard representation of the data across the views. Should each view be left in full color, there is an immediate and obvious bias to a particular color per view that isn't practically useful.

3.2 Sample Extraction

Each HPTLC view image was provided, as seen in Figures 2.1-2.3, where all 24 samples were placed adjacently and saved altogether in one image. However, in order to extract information for each sample, we individually separated every sample within a view, repeating this for all views.

3.3 Sample Vectorization

Once each sample was able to be handled individually, they were all transformed into a single vector. This was done by taking the average value of a row of the sample and creating a vector of each row's average value.

Figure 3.1 is an example of what this looks like, however each sample has been widened from a one-wide pixel for the sake of visibility.

View Index = 2

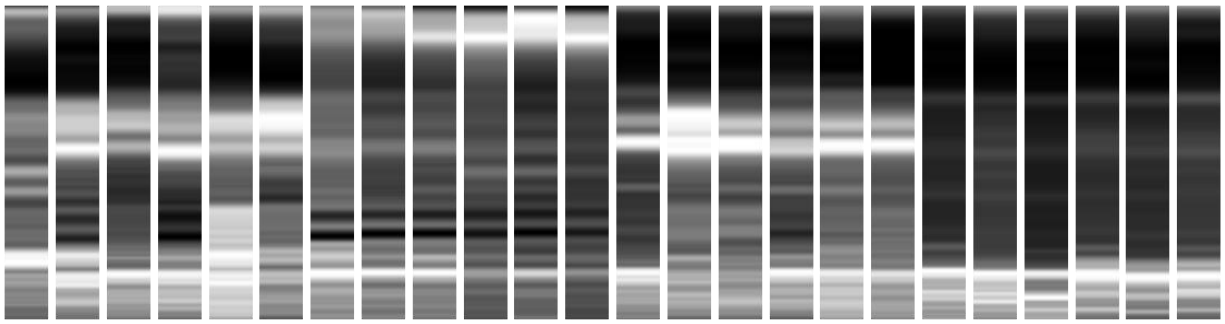

Figure 3.1: Widened 1-Dimensional Sample Vectors

SECTION 4

MOTIVATIONS FOR MULTIPLE VIEWS

Using multiple views adds a layer of complexity and possible confusion when trying to implement and comprehend the various algorithms, so we provide the reasoning as to why we decided to make use of all the views.

4.1 Dataset Size

First and foremost, note that a single view has as few as three samples per class. Although the data itself may be relatively straightforward to analyze, it has long been determined that generally, the larger the training sample size, the more accurately a model will perform (S. J. Raudys 1991). Given the small class size, we decided to test our models on each view to gain the most insight into the data possible. We did not know whether one specific view would perform better than another, or perhaps they might all perform similarly. As small sample size is bound to produce a high variance, testing each view at least allows for some generalizable understanding of the task rather than relying on a singular model's performance on one view (Brownlee 2020 "Impact-").

4.2 Generalizability of Algorithms

Using multiple views enables us to eliminate the hypothesis that a single view is working particularly well or badly within a model, and it also gives evidence for the performance of the applied algorithms. Again, rather than a feature selection technique being validated in use with just one view, we can provide more generalized evidence of its effectiveness.

4.3 Multi-view Learning

Multi-view learning uses various techniques to combine data from multiple views about a sample of data to perform machine learning tasks. Specific implementations vary, but the core of the idea is that we can measure information about a sample in different ways. For example, “in web page classification, there are often two views for describing a given web page: the text content of the web page itself and the anchor text of any web page linking to this web page.” (Zhao 2017). In our case, the different measuring techniques vary based on the conditions the images were taken, resulting in our eight views.

We could use these multiple views to perform feature extraction in ways we couldn’t if we only used one. For example, we can naively combine all the views and extract the most relevant features across all views combined using a model. This and other strategies that necessitate the use of multiple views were performed and will be discussed later.

SECTION 5

METHODS FOR FEATURE SELECTION AND RULE EXTRACTION

As a brief reminder, the goal for us is to select a set of features from our dataset that we can use to extract a simple set of rules so that the problem can be better understood. To this end, there were quite a few computational methods used to remove the unnecessary features and select the most useful ones before determining the rules. This section will discuss the breadth and sequence of those that we used.

Figure 5.1 displays a summary of each of these steps, the data input into the step, and the data output from the step.

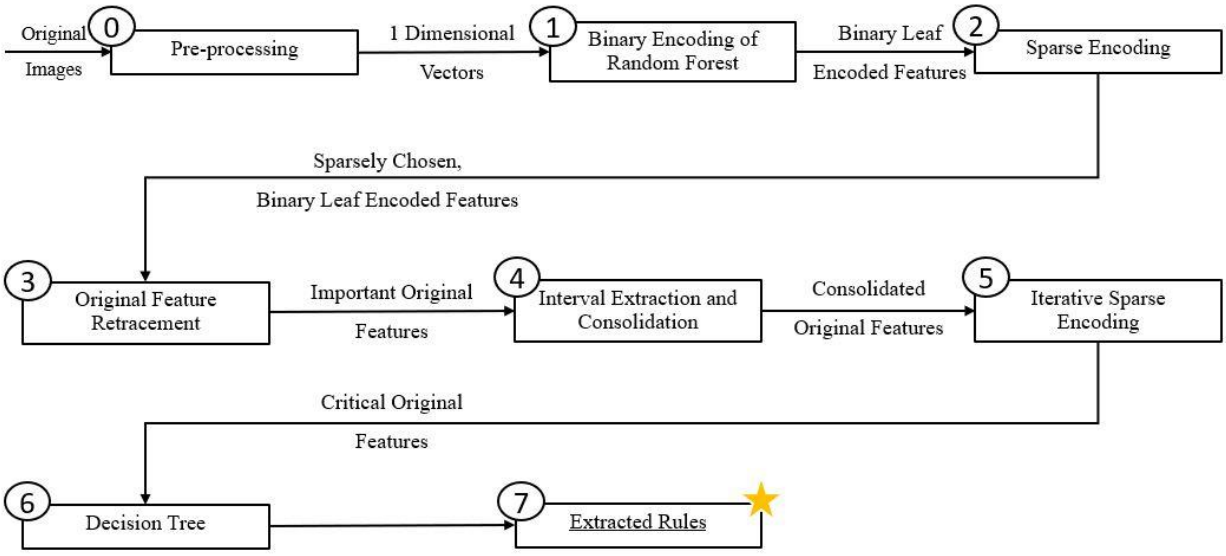

Figure 5.1: Chain of Operations for Feature Selection and Rule Extraction

5.1 Binary Encoding of Random Forests

5.1.1 Overview

The first step after preprocessing the data is creating a Binary Encoding of a Random Forest. Abstractly, this "maps a sample point to a space defined by the entire set of leaf nodes (rules) of the forest" (Liu 2012). This method was performed on each view separately, generating a Binary Leaf Encoded Feature space for each view. While the following explanation indicates how this encoding is done on one view, note that it was indeed performed on each of them.

5.1.2 Motivation and Method

The Binary Encoding of a Random Forest is achieved by first training a Random Forest (RF). An RF is an ensemble of Decision Trees, which themselves are models used for prediction. Decision Trees choose features from a sample to create a model structured like the tree data structure. At each node, there is a value with a threshold representative of a feature. A decision is made about whether a sample should be placed in the subset of one group, or another based on whether the value for this sample at this feature falls below or above the threshold. Each sample has a path from the root node to a leaf which categorizes it. In our case, the leaves represent one of the five classes. This chain of decisions made at each node along the sample's tree traversal can be interpreted as a set of rules. This chain of rules is easy to interpret as there is a clear sequence of steps, and the feature impact at each node is stated (Liu 2012).

As mentioned, an RF is an ensemble of these decision trees, so it also provides these decision rules. The key difference is that there is a much larger number of them. However, in an effort to select only the most critical of the features and rules, we performed a Binary Encoding of the RF. The Binary Encoding of our samples is achieved as follows. For each sample passed

into our trained RF model, we will create a vector of length equal to the number of leaves in the forest. This vector's value will be in the order of the leaves in the forest. Finally, a particular leaf node's value in this vector is set to 1 if the sample reaches that leaf and 0 otherwise. This transformation creates a binary vector for each of our samples. Each binary vector holds information about the decision path for a sample because it "defines which rules are active" (Liu 2012). Active rules are useful as they can be used to determine the most critical features from our original data. This will be done later after further feature elimination.

5.1.3 Creation of [Tree: Leaf Count] Dictionary

After training an RF, we created a dictionary for the forest. Each key in the dictionary is a tree object, and each value is its number of leaves. Furthermore, it should be noted that the tree keys were ordered as they were in the RF. This map is immediately used to create the binary vector for each sample. However, it is also used in step 3 to determine the original features used in the decision paths of the trees.

5.1.4 Summary and Results

As a reminder, the samples fed into this step were the preprocessed images. We input 1 Dimensional vectors for 24 samples with 1030 features for each view. The resultant binary leaf encoding resulted in counts of features between 99 and 123. These vary as each view has its own RF and, thus, its own number of leaf nodes. The specific view and feature count association can be seen in Table 5.1.

Table 5.1 View to Feature Count after Binary Encoding of Random Forest

View #	Feature Count	View #	Feature Count
1	114	5	120
2	109	6	100
3	113	7	99
4	123	8	103

5.1.4 Note on Random Forest Model

While the training of the Random Forest was mentioned, the details of this were glossed over. We would like to note that the optimization of the parameters of the RF was important. However, it is not crucial to understanding the Binary Encoding of Random Forest, so the discussion for these optimizations is set aside for Section 6.

5.2 Sparse Encoding for Rule Extraction

5.2.1 Overview

As mentioned, the Binary Leaf Encoded Features from the previous step can be useful for determining the most active features. This step is where they serve that purpose. We use binary encoding as an input into a particular Support Vector Classifier (SVC), which functionally selects the most important features from it. The SVC we use is a 1-norm Support Vector Machine, which uses “l1” normalization. This configuration excels at handling inputs with “redundant noise features” (Zhu 2003). Abstractly, it assigns a weight to a feature where the

higher the weight, the more important the feature is. This means that, with appropriate tuning, this type of SVC will set certain features' weight to 0, deeming them unimportant with regard to a final, simplified model. The tuning for how aggressive the model should be with setting features to 0 is determined by the regularization parameter. This parameter will be discussed later. The behavior where unnecessary features are eliminated ultimately enables feature selection.

Finally, as was stated in the prior section, this transformation was applied to all views individually, even though it is described as only being applied to one.

5.2.2 Motivation and Method

With an understanding of how the SVC generally works, we now apply it to the Binary Leaf Encoded Features. As a reminder, these encoded features represent the decision rules found in the Random Forest. This means a 1-norm SVC machine trained on the encoded features will eliminate the least important decision paths as it determines a sparser feature space. This forward feeding technique works well as it is not simply training multiple models on the same set of features. Instead, it intuits information about the original feature space at each step. The Binary Leaf Encoded Features are a mapping of the features into decision rules, while the Sparse Encoded Features are a selection of those mapped decision paths most important in determining a sample's classification. Thus, features that are not used in the SVC can be ignored as decision paths as they presumably don't have an impact on the decision rules extracted from the Random Forest (Liu 2012). The extraction of the original features from the "Sparsely Chosen, Binary Leaf Encoded" features is performed in Step 3.

5.2.4 SVC Tuning: Iterative Optimization without Cross-Validation

With the goal of producing a sparser feature set, the regularization parameter was tuned on the SVC model. With fewer features used, we can eventually make a simpler model. However, fewer features can also mean a less accurate model. With the desire to maintain accuracy, we noted the minimum number of features required to produce 100% accuracy on a view. This required iteratively training the model on a range of regularization parameters and noting the first occurrence of 100% accuracy. While the originators of the Sparse Encoding technique recommend finding this regularization parameter using cross-validation, our dataset was too small to be able to perform the operation reasonably (Liu 2012). As such, we used simple iteration where many models were trained with various parameters until we noticed the minimum number of parameters necessary to reach the asymptotic 100% accuracy.

5.2.3 Note on Training Data and Configuration: One v Rest

Although we claim 100% accuracy of classification for each view, and this is accurate, the classification was performed on the same training data. Due to the incredibly low count of samples per class (only three in some cases) it wasn't feasible to divide the data into training and validation. Fundamentally, the concern of the veracity of the presented accuracy values is that the model is overfitted to the inputs. However, this effect is mitigated in two key ways.

First, given the nature and tuning of the 1-norm SVC, we explicitly attempted to create a model that was as sparse as possible while still hitting an accuracy metric. The use of sparse support vector machines has been noted to reduce overfitting by other researchers in the past (Han 2014). Second, we used One-versus-Rest (OvR) training. OvR is a method of training a multi-class classifier by training multiple binary classifiers, one for each class. Each binary

classifier is trained to distinguish the samples of its corresponding class from the samples of all other classes. This can help prevent overfitting because it allows the SVM to focus on one class at a time rather than trying to classify all classes simultaneously.

Regardless, the prescribed accuracy isn't ideal, but we believe it to be sufficient given the model's design.

5.2.5 Summary and Results

This step of the operations chain, "Sparse Encoding for Rule Extraction," consumed the Binary Leaf Encoded Feature vectors from the previous step. Each view held 24 vectors, one for each sample. Each vector was about 110 bits long, varying by the view. Each view was trained using a range of regularization parameters until a parameter was found that produced 100% accuracy with the fewest features used. The results of the remaining number of features in each view can be seen in Table 5.2. As a reminder, this feature count is a count of the useful decision paths per view, not the number of original features. That count is determined in the next step, Original Feature Retracement.

Table 5.2 Remaining Count of Features per View after Sparse Encoding

View	Accuracy	Remaining Feature Count (of Binary Leaf Encoding)
1	100%	33
2	100%	24
3	100%	20
4	100%	38
5	100%	17
6	100%	22
7	100%	23
8	100%	21

5.3 Original Feature Retracement

5.3.1 Overview

After the decision paths represented by the Binary Leaf Encoded Patterns have been trimmed down to only the most relevant, we now map back these decision paths to the features used within them. Again, this was done for each view individually.

5.3.2 Method

First, the tree objects in all the Sparsely Chosen, Binary Leaf Encoded Features from step two were gathered. Each list of trees was ensured to be only one copy of a tree, removing duplicates when necessary. This mapping was performed using the dictionary of Tree Object-to-Leaf Count created in step one. After determining all of the unique trees, representative of the

leaves in the binary encoded leaves, each of the tree objects was used to determine the original features present at each of their nodes.

5.3.2 Summary and Results

We determined the original features used in the trees whose leaves were retained after the sparse encoding of the previous step. This step consumed the Sparsely Chosen, Binary Leaf Encoded Features and produced a small set of Important Original Features. The counts of original features remaining in each view can be seen in Table 5.3.

Table 5.3 Count of Important Original Feature by View

View	Important Original Feature Count
1	76
2	64
3	55
4	91
5	65
6	48
7	45
8	58

5.4 Interval Extraction and Consolidation

5.4.1 Overview

After the previous step, the original plan was to move on to what is now step 6. However, in observing the output of the previous step, we noticed a pattern in the Important Original Features. Figure 5.2 shows a list of the Important Originals for one view.

```
[  8  10  21  68  75 105 112 151 191 219 226 232 236 257 265 273 276 279
 280 284 339 346 347 349 351 358 370 379 384 385 391 392 393 401 404 415
 432 486 488 492 497 498 505 509 520 556 580 585 609 632 634 655 661 665
 667 669 676 683 695 696 730 731 761 787 801 813 826 848 854 859 860 874
 876 908 910 920]
```

Figure 5.2 Important Original Features in View 1

Our observation of the above feature space is that the chosen features seemed to occur in groups. There are several nearly continuous values followed by a massive gap of unused values before the next clump of important features. We can visually understand this by visually inspecting a sample. See Figure 5.3 for an example.

Figure 5.3 Widened 1-Dimensional Vector of Sample 2 in View 1

A sample is fundamentally composed of bands of values. These bands represent the separated components within whatever substance that is being analyzed. The varying color of the bands helps researchers analyze samples even without computational methods. Thus, it is reasonable to conclude that the model also uses bands to classify the samples. However, due to the nature of the Random Forest and how nodes are picked in each of its Decision Trees (randomly), what a human might call a "single band" can be represented by a range of features. After all, even though two separate pixel values may be equivalent due to being adjacent portions of the image, they are in different pixels and, thus, different features. Furthermore, this behavior reinforces the usefulness of the decision path encoding created by the Random Forest. We now see that many paths use the same feature (a specific single band) to determine the sample's classification.

Given this analysis and our goal of extracting only the most valuable features, we consolidated each clump of features into a single value.

5.4.2 Method

Consolidating the features in Figure 5.2 {Important Original Features in View 1} begins with determining the interval a clump of features covers. This requires some tuning. Technically, values are not continuous if they are not adjacent. However, based on our analysis, we can intuit that having a small gap between the selected features to still consider them "continuous." As such, we created an algorithm to construct an array of intervals from the array like that in Figure 5.2 {Important Original Features in View 1} with a tunable parameter called the Relaxation Value. The Relaxation Value determines the maximum distance apart that two features can be and still be considered part of a single interval.

Various Relaxation Values were tested, and their performance was gauged by the count and compactness of their intervals. We desired a low count of intervals balanced with intervals that did not span too large of a range. Ultimately, a Relaxation Value of 10 was used. Values below 10 often produced intervals that could reasonably be joined further. Values above 10 either had no significant impact on the count of intervals or, in the extremely high Relaxation Value Cases, produced too few intervals that covered far too large of a range. Finally, should a value be "alone" (not within a range) it was ignored.

5.4.3 Determining the Center Value

Once we determined each possible interval, we calculated the center feature within that range and used that to replace all previous features within the interval. There was an alternative approach to determining the best representative feature of an interval, and it will be discussed in Section 7. However, the alternative approach was not used due to its increased complexity and lack of improvement over the stated approach.

5.4.4 Multiview Learning

Although Figure 5.2 {Important Original Features in View 1} shows an example of a single view's Important Original Features, this is not the final array we used to determine the intervals. Instead, we combined each view's Important Original Features into one long sorted list. Then, the intervals were extracted from this list. This means that each view's feature set is now equivalent.

This is the step in the chain of operations that makes use of the idea of Multiview learning. Another method was explored but not implemented, and it will be discussed in Section 8. Beyond just making use of the multiple views, this decision was made for the following

reason. When all of the view's Important Original Features were concatenated and sorted into a single array, many features were duplicated a number of times over. This means that they already existed in multiple views. Furthermore, some ranges became even more defined, with more intermediary values present. These observations motivated us to conclude that using every view would allow for more accurate and higher-resolution intervals.

5.4.5 Summary and Results

The Interval Consolidation and Extraction performed in this step consumed the Important Original Features and compacted them into a single set of Consolidated Original Features. The counts of the Important Original can be seen in Table 5.3 {Count of Important Original Feature by View}. The count of features after the Interval Consolidation and Extraction is 12 across all views. A visualization of these twelve features plotted on View 1 is provided in Figure 5.4. Each horizontal red line is a feature.

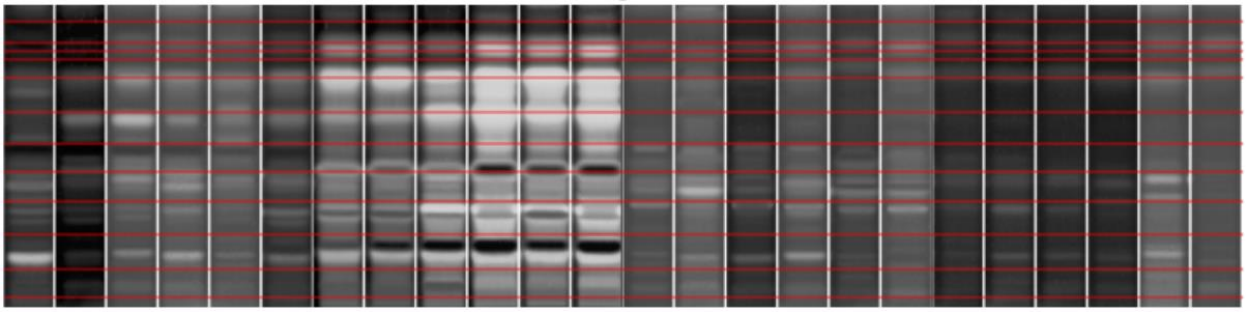

Figure 5.4 Consolidated Original Features Plotted on View 1

It can easily be seen that many of these features do line up with the horizontal dark or light bands present in some classes of the samples.

5.5 Iterative Sparse Encoding without the use of a Random Forest for Rule Extraction

5.5.1 Overview

We can now perform a Sparse Encoding on the original features. The process and logic in this step are identical to step 2. We use a 1-norm SVC on the input features, "Consolidated Original Features," in this case, and eliminate features that are assigned a weight of 0 by the model.

5.5.2 Motivation & Method

The original paper, which discussed using binary leaf-encoded vectors in rule extraction using sparse encoding, mentions an iterative approach. Their suggestion was to use the output from the sparse encoding as the deciding factor for the input of a new Random Forest, thus iterating through the process again (Liu 2012). While this is not what we do here, it inspired the use of another iteration using sparse encoding. The key difference is that our sparse encoding is performed on a set of the original features directly, not being passed back through a Random Forest to generate binary vectors again.

A key difference as compared to the previous implementation of the sparse encoding method is that we did not expect 100% accuracy for each view this time. Instead, we sought a high accuracy.

5.5.3 Summary and Results

As a brief synopsis, this step consumes the Consolidated Original Features, performs sparse encoding using a 1-norm SVC, and produces "Critical Original Features." The results can

be seen in Table 5.4. Ultimately, this step did not reduce the number of features by as significant a proportion as previous steps, but reducing the space even further is helpful, nonetheless.

Table 5.4 Count of Critical Original Features after Iterative Sparse Encoding

View	Accuracy	Critical Original Feature Count
1	100%	11
2	100%	9
3	87.5%	9
4	100%	9
5	91.7%	10
6	100%	10
7	100%	11
8	95.8%	11

5.6 Decision Tree Rule Extraction

5.5.1 Overview

Now that the most critical features have been determined, reducing the feature space from 1030 to ~10, we can use these features to determine a set of simple rules. As described in Section 5.1.2, Decision Trees are excellent at creating a set of easily interpretable rules, which is what we do in this step.

5.5.2 Method and Outcome

A Decision Tree was trained for each view, using its own set of Critical Original Features. The model parameters were tuned to provide 100% across all views. Each model produced a Decision Tree. These trees can be viewed in Figures 5.5-12.

5.7 Conclusions

This process of feature selection and rule extraction functionally reduced a space of 1030 features to around 10. It allowed us to produce a simple decision tree for each view. The particulars for the "meaning" of each rule in each tree are not relevant to the scope of this paper, but there are a few observations to make about the produced trees.

Most obviously, the trees are not equivalent. Each view has a unique tree. However, there is some similar behavior across the trees. For example, when any single sample from class 2 is separated, all the class samples are also divided into subgroups. Even when comparing different features, class 2 excellently differentiates itself from the other classes. Finally, while some trees do have a more complex decision path to a particular leaf, such as in View 4, View 7's decision path per class is incredibly simple. It separates nearly every class into its own subset with one decision. View 8 also nearly accomplishes this. Perhaps the form of imaging represented by these views produces the most reliable and readily interpretable results.

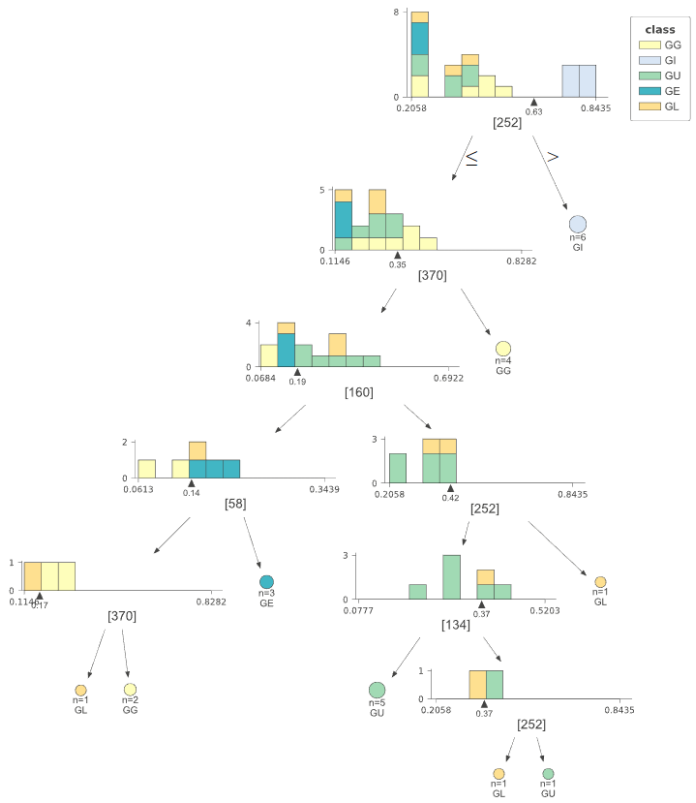

Figure 5.5 Extracted Rules in Decision Tree of View 1

Figure 5.6 Extracted Rules in Decision Tree of View 2

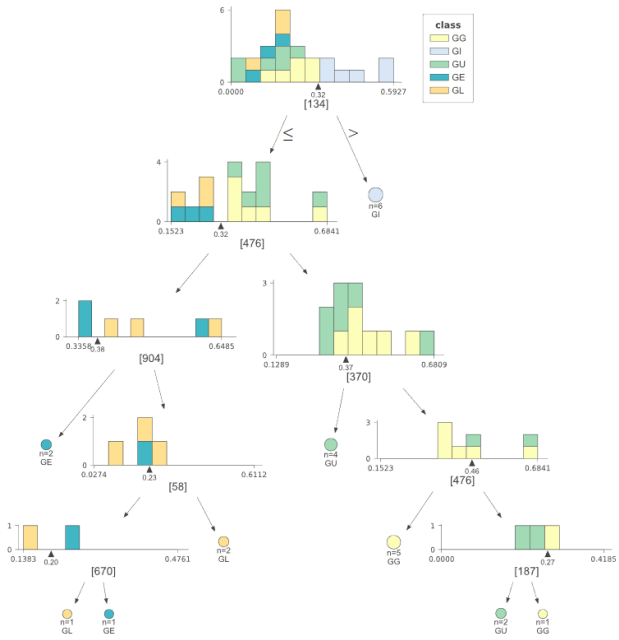

Figure 5.7 Extracted Rules in Decision Tree of View 3

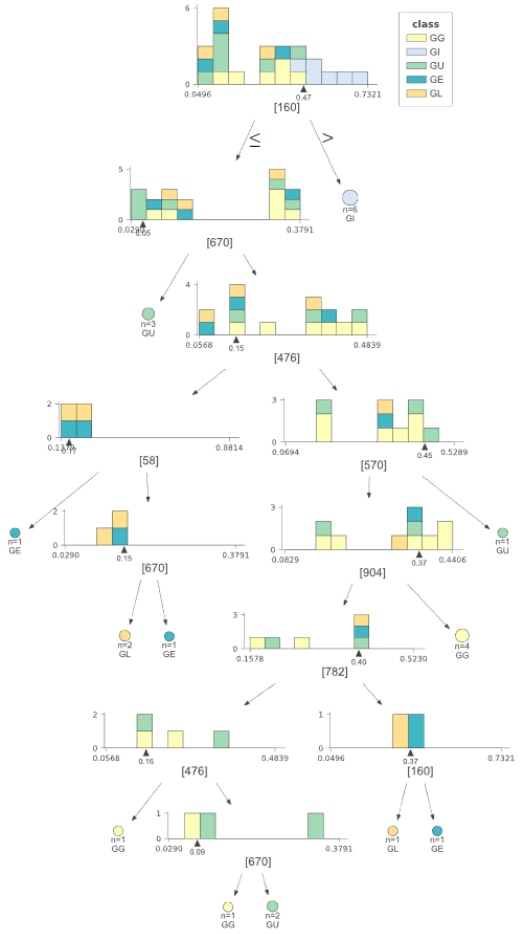

Figure 5.8 Extracted Rules in Decision Tree of View 4

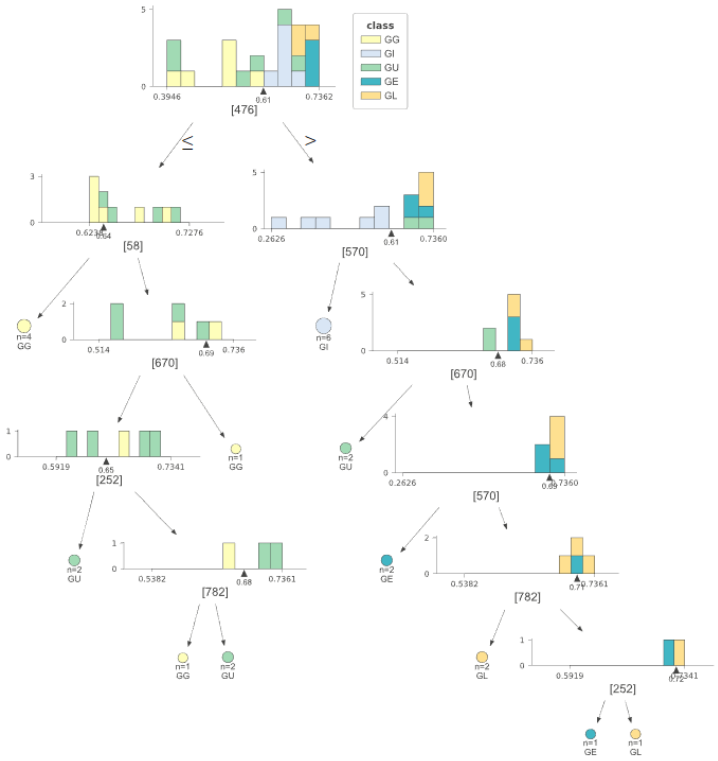

Figure 5.9 Extracted Rules in Decision Tree of View 5

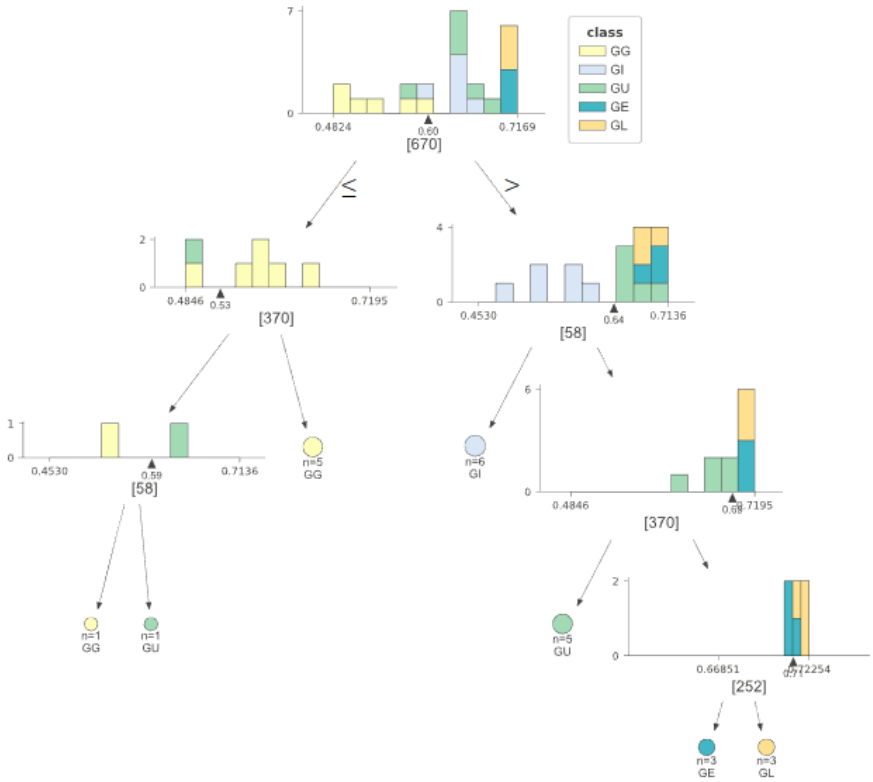

Figure 5.10 Extracted Rules in Decision Tree of View 6

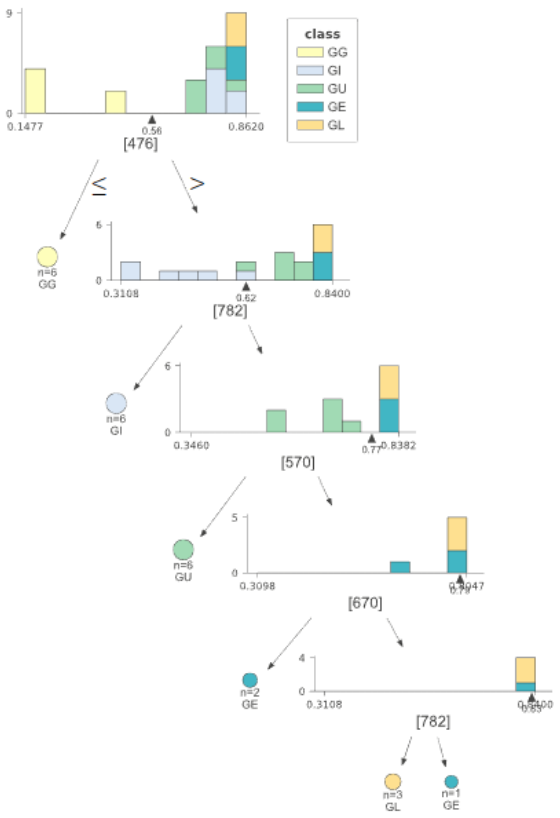

Figure 5.11 Extracted Rules in Decision Tree of View 7

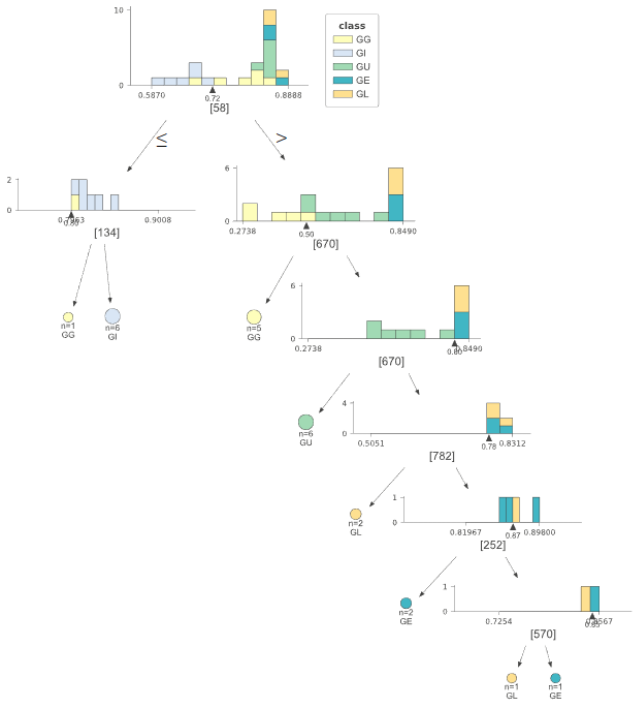

Figure 5.12 Extracted Rules in Decision Tree of View 8

SECTION 6

RANDOM FOREST PARAMETER OPTIMIZATION

6.1 Objectives

As mentioned in Section 5.1.4, efforts were taken to optimize the Random Forest model used for the Binary Leaf Encoding, which will be discussed in this section. Primarily, our key objective was to have the fewest number of original features represented in our Random Forest model. Of course, similarly to the 1-norm SVC model, the fewer features were present, the lower the accuracy of the RF. Therefore, we sought at least a near-perfect accuracy from our model while tuning its parameters. This means miss-classifying no more than 3 samples, preferably fewer. Regarding the Random Forest model, this specifically means that we desire the fewest nodes present across all our trees.

6.2 Outcomes

To this end, we conducted various tests of our model. The two key parameters to optimize around are the maximum depth of each tree and the number of trees in the forest. Using these values, you can calculate the maximum number of nodes present in the forest. Simply find the maximum number of nodes in a binary tree using the maximum depth (h): $2h - 1$. Then, multiply this value by the number of estimators. The result is the maximum number of nodes in your forest.

Keeping the criteria in mind, we performed a number of tests on forests with different depths and numbers of estimators. When trees were forced to depths less than three, they did not

reach high enough accuracy without a very high number of estimators. At a max-depth equal to three, we achieved relatively good accuracy with a relatively low number of estimators. Tables 6.1 and 6.2 show the accuracy for each view for a Random Forest with a max depth 3 and a number of estimators equal to 30 and 40, respectively.

Table 6.1:
Accuracy across All Views for Random Forest with Max Depth: 3, N_Estimators: 30

View	Accuracy
1	95.8%
2	100%
3	100%
4	87.5%
5	95.8%
6	100%
7	100%
8	100%

Table 6.2:
Accuracy across All Views for Random Forest with Max Depth: 3, N_Estimators: 40

View	Accuracy
1	100%
2	100%
3	100%
4	95.8%
5	100%
6	100%
7	100%
8	100%

However, when the max depth was set to 4, we were able to achieve comparable accuracy with even fewer estimators. See Table 6.3.

Table 6.3: Accuracy across All Views for Random Forest with Max Depth: 4, N_Estimators: 20

View	Accuracy
1	95.8%
2	95.8%
3	100%
4	91.7%
5	95.8%
6	100%
7	100%
8	100%

After calculating the maximum number of nodes in all three of these forests, we get the results in Table 6.4.

Table 6.4 Maximum Number of Nodes in each RF Model

Model	Max. Number of Nodes
Max Depth = 3, N_Estimators = 30	150
Max Depth = 3, N_Estimators = 40	200
Max Depth = 4, N_Estimators = 20	140

Because our objective is to have the fewest possible nodes, we used the model with a max depth of 4. Finally, a brief note that while this calculation shows the maximum number of

nodes, it is unlikely that this number is actually reached as Decision Trees do not often fill out their full binary structure. The model we used had fewer than 140 nodes.

SECTION 7

ALTERNATIVE IMPLEMENTATION FOR DETERMINING AN INTERVAL'S CENTER

7.1 Motivations

As discussed in Section 5.4.3, there was an alternative implementation for determining the center value of an interval. As a reminder, the final implementation took the numerical center between the two ends of the interval. This calculation is (End of Interval) + (Start of Interval) / 2. Although the intervals are built from many values, it is very often the case that the selected center value is not present in the semi-continuous values used to define the interval. This begs the following question. Would using a value present in the set used to construct the interval generate better results?

7.2 Implementation

At its essence, the alternative implementation chooses the median value of all the values used to construct a particular interval. It performs this by keeping track of all the values used to construct an interval. Then, we pick the center feature from this ordered list. This guarantees that we will use a feature that was produced by the Sparse Encoding and the Binary Encoding of the Random Forest steps.

After extracting and consolidating the intervals, we used the resultant features as we did in the final implementation and put them through another SVC for a round of Iterative Sparse Encoding.

7.3 Results

The median value extraction method was compared to the mean value extraction method across all views based on resultant feature count and accuracy.

Regarding the feature count, the median implementation performed as well or better than the mean implementation. However, this difference is negligible as it is by one to three features at most.

In terms of accuracy, the two extraction methods performed better than each other on different views. However, if we calculate the average accuracy value across all views, the mean method scores a 96.9% while the median method only scores a 94.8%. The mean method is slightly more accurate than the median method in our test.

Ultimately, we determined that the implementations produced such similar results that the difference between them was not significant enough to warrant using one over another. Thus, we decided to use the simpler implementation. That is simply choosing the center value of an interval.

Table 7.1
Mean Center Interval Extraction

View	Accuracy	Feature Count
1	100%	11
2	100%	9
3	87.5%	9
4	100%	9
5	91.7%	10
6	100%	10
7	100%	11
8	95.8%	11

Table 7. 2
Median Center Interval Extraction

View	Accuracy	Feature Count
1	100%	10
2	100%	9
3	91.7%	7
4	91.7%	9
5	83.3%	10
6	95.8%	7
7	100%	8
8	95.8%	8

SECTION 8

POSSIBLE EXTENSIONS AND FUTURE WORK

The following items are a few ideas related to the data or processes used in this project which could be further explored.

8.1 Multiview Binary Leaf Encoding

In the demonstrated method, each view individually generated a Binary Leaf Encoding after preprocessing the data. However, it is possible to perform a very simple multiview training technique where each view is concatenated so that they all functionally become one dataset. This concatenated superset could then be used to generate a Binary Leaf Encoding and continue through the remaining steps described previously. The one other exception in the series of steps would be that when one performs Interval Consolidation and Extraction, they will not join multiple views together to extract intervals.

Multiview feature selection can increase accuracy (Komeili 2021). Therefore, an approach where this is incorporated from the start might allow for the same high standards of accuracy with fewer features. Although we eventually use multiple views in our process, this approach would leverage the benefits from the very beginning.

8.2 Increased Iterations

As mentioned in Section 5.5.2, the original paper, which discussed using binary leaf-encoded vectors sparsely encoded for rule extraction, mentions an iterative approach. Their suggestion was to use the output from the sparse encoding as the deciding factor for the input of

a new Random Forest, thus iterating through the process again (Liu 2012). We did not do this, so it is certainly a path that could be explored for further research.

Furthermore, while the "natural" step for this, as envisioned by the original authors, would be after our step 3 "Original Feature Retracement," it could just as easily be performed after step 4 "Interval Extraction and Consolidation." Either of the outputs of these steps could reasonably be used as the input for another iteration of Binary Encoding of a Random Forest and Sparse Encoding.

8.3 Applying Interval Extraction and Consolidation on Higher Dimensional Data

Finally, in terms of very different datasets, we could imagine use cases where something similar Interval Extraction and Consolidation is applied to higher dimensional data. Of course, this would require an entirely different implementation, but choosing a single representative for a cluster of features could produce useful results, especially in cases where the goal is to select features for rule extraction.

BIBLIOGRAPHY

Brownlee, Jason. “How to Choose a Feature Selection Method for Machine Learning.” *MachineLearningMastery.com*, Machine Learning Mastery, 20 Aug. 2020, https://machinelearningmastery.com/feature-selection-with-real-and-categorical-data/.

Brownlee, Jason. “Impact of Dataset Size on Deep Learning Model Skill and Performance Estimates.” *MachineLearningMastery.com*, 25 Aug. 2020, https://machinelearningmastery.com/impact-of-dataset-size-on-deep-learning-model-skill-and-performance-estimates/.

Han, Henry, and Xiaoqian Jiang. “Overcome support vector machine diagnosis overfitting.” *Cancer informatics* vol. 13,Suppl 1 145-58. 9 Dec. 2014, doi:10.4137/CIN.S13875

Liu, Sheng et al. “Combined rule extraction and feature elimination in supervised classification.” *IEEE transactions on nanobioscience* vol. 11,3 (2012): 228-36. doi:10.1109/TNB.2012.2213264

M. Komeili, N. Armanfard and D. Hatzinakos, "Multiview Feature Selection for Single-View Classification," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 43, no. 10, pp. 3573-3586, 1 Oct. 2021, doi: 10.1109/TPAMI.2020.2987013.

Marathe, Written by Sandesh. “HOW HPTLC WORKS: A Guide for Researchers at the Bench.” *Bitesize Bio*, 11 June 2020, https://bitesizebio.com/47784/hptlc-basics-and-instrumentation/.

S. J. Raudys and A. K. Jain, "Small sample size effects in statistical pattern recognition: recommendations for practitioners," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 13, no. 3, pp. 252-264, March 1991, doi: 10.1109/34.75512.

Sathyajith, Deepthi. “High-Performance Thin Layer Chromatography (HPTLC) in the Pharmaceutical Industry.” *NewsMedical*, News Medical, 17 Oct. 2019, https://www.news-medical.net/life-sciences/High-performance-Thin-Layer-Chromatography-(HPTLC)-in-the-Pharmaceutical-Industry.aspx#:~:text=The%20pharmaceutical%20industry%20has%20witnessed,of%20bulk%20drugs%20and%20formulations.

Talari, Saikumar. “Random Forest vs Decision Tree: Key Differences.” *KDnuggets*, 2022, https://www.kdnuggets.com/2022/02/random-forest-decision-tree-key-differences.html.

Zhao, Jing, et al. “Multi-View Learning Overview: Recent Progress and New Challenges.” *Information Fusion*, vol. 38, 2017, pp. 43–54., https://doi.org/10.1016/j.inffus.2017.02.007.

Zhu, Ji, et al. ‘1-Norm Support Vector Machines’. *Advances in Neural Information Processing Systems*, edited by S. Thrun et al., vol. 16, MIT Press, 2003, https://proceedings.neurips.cc/paper/2003/file/49d4b2faeb4b7b9e745775793141e2b2-Paper.pdf.