An Algorithm for the Filtering and Classification of CSI Data-sets

Ryan Lanciloti
Email: ryanjl9@iastate.edu

November 12, 2021

Contents

1 Preface **2**

2 Background **2**
2.1 The End Goal (a.k.a The Problem that Needs a Solution) 2
2.2 The Current Approach 2
2.3 The Problem (Or What I Think the Problem is) 3

3 The Algorithm **6**
3.1 Introduction 6
3.2 Getting the Variance of Our Data 6
3.3 Classification (a.k.a The Hard Problem) 7
3.4 Conclusion/Summary 9

4 Approaches to Classification **10**

5 Short Comings **10**

6 Conclusion **11**

1 Preface

The purpose of this paper is to articulate the methodology behind an algorithm that I, Ryan Lanciloti, have devised to increase the accuracy of state classification in the system described later on in the paper. By the end of the paper, you, the reader, should have an understanding of what the over-arching problem is, why we haven't solved this problem as of yet, what my algorithm does, how it's supposed to solve the over-arching problem, how it might fail to solve the over-arching problem, and why I don't of a good way to implement this algorithm.

2 Background

2.1 The End Goal (a.k.a The Problem that Needs a Solution)

What is it that we are trying to do in the first place? We want to develop a system that allows us to classify the current state of a door. Another way to phrase the problem statement is that we want to use CSI data to determine the current position of a door. The important points here are the mention of **CSI** and the mention of **state/position**.

What is CSI? CSI, also known as Channel State Information, is a property of WiFi radio waves which provides us with information about the path a wave took to reach a receiver. How it works is that there are a certain number of sub-carriers associated with a given transmitter, for our system, it's 53 sub-carriers, and each sub-carrier relays it's phase and magnitude when it reaches the receiver. The theory is that as a wave propagates through a room, it will bounch off of the objects in said room. As it does this, it will cause a phase shift and a change in overall magnitude. With 53 different sub-carriers, each reflecting off of the objects in a room in different ways, we can use this data to do basic object detection.

The idea of "door state" is simple in concept, however I feel the need to provide a concrete definition as to what it means in relation to our problem. Door state refers to the position of a door with relation to it's frame, and it's extends past a simple **open** and **closed**. When I state that we want to detect door state, I mean to say that we want to determine that approximate angle at which the door is currently open to. We use 7 discrete values, 0°, 15°, 30°, 45°, 60°, 75°, 90°.

So to restate the problem statement one last time, essentially we want to use WiFi wave propagation throughout a room to determine the angle at which a door is open to with relation to it's frame.

2.2 The Current Approach

Our current setup is as follows: There's a sender ESP32 and a Receiver ESP32, the sender is transmitting the WiFi radio waves and the receiver is listening for these waves.

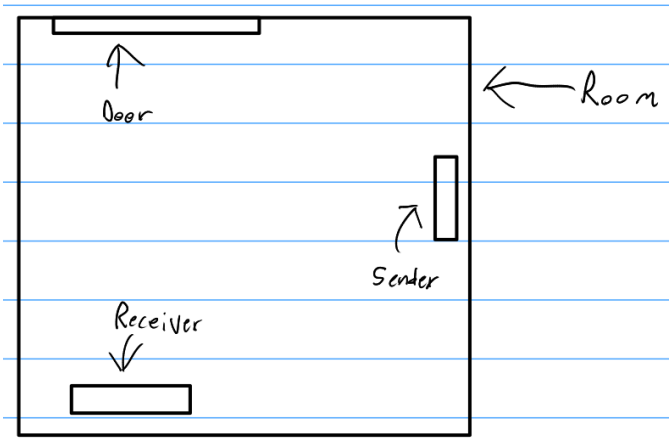

Figure 1: Environment Setup

In figure 1, there is a basic diagram of the setup. Real simple, there's a door and the two ESP32s. Once the receiver receives the radio wave, it extracts out the phase and magnitude from the 53 sub-carriers and sends this information off to a back-end. The back-end is comprised of a flask server acting as an API and a machine learning model. The flask server passes this information on to the machine learning model to do inference on the data which is collected. This inference produces a confidence level for each of the 7 discrete possible states which tells us the percent chance it's in any given state.

To train the model, we build up a training data-set. The general flow for building this set is as follows: we open the door up to one of the discrete degree markers, we gather points for 3 minutes, and then we move to the next discrete degree marker. After labeling each data point in the training set, (i.e. we label all the data points for 0°, 15°, etc.) we feed this training set into the model to fit the data-set. The theory is that when the door is in each of it's discreet states, the incoming data should match that of the training data, and we should therefore be able to determine the current door state.

2.3 The Problem (Or What I Think the Problem is)

In order for the above approach to work, the data at each discreet degree marker must be unique with regards to the other discreet degree markers. The theory behind CSI allows us to infer that as objects in a room change, some portion of 53 sub-carriers should reflect this change. So what happens in our system? When the system is at rest, we experience a high amount of fluctuation in the output of our machine learning model. One second it thinks the closed door is open to 75°, the next it thinks it's open to 15°. Why do we see such high volatility in the output of our model inference if the system isn't changing?

To answer this question, we have to look a little more closely at our model. We are using a Deep Neural Network, DNN for short. A DNN is comprised of a series of nodes, each with weights and biases, with multiple convolutional layers that do some sort of mathematical

calculation on an input value and returns the output. Each layer acts as the input to the next with the first layer receiving the raw CSI data as input. To train a DNN, you pass in a data set with a known expected output and depending on the actual output of the DNN, you adjust the weights and biases of the nodes in each convolutional layer to provide a more accurate guess. For our system, the theory is that if we pass a 100 points data points, each for 0°, the DNN should learn to identify these data points by their **statistical similarity**. In our system, each data point has 106 fields (53 phase and 53 magnitude) associated with it.

There ends up being two main issues with the data-set, lack of distinction between the degree marker sets and high volatility within each of the degree marker sets. For lack of distinction, let's look at genomes. According to this article, humans share approximately 99% of their DNA with gorillas. If we were to train a DNN to identify whether something was human or a gorilla by feeding it a genome, we'd find that we'd have a very difficult time increasing our accuracy if we feed it the whole genome. This is because 99% of the data is exactly the same and therefore, it would be unlikely that the model would find any statistical difference between the two. For a more tangible example, take the real Monalisa painting and a fake Monalisa painting. If the only difference between the two paintings is that the fake uses the wrong shade of green in the background, any random person would have a hard time identifying that. However, if we cut out only the backgrounds of both and compared them, it would be significant easier to identify difference between the two paintings. Looking back to the genome example, if we stripped out portions of the two genomes that we know are going to be exactly the same, we'd see a higher percentage of distinction between the human and gorilla genomes. This means that our DNN would have an easier time identifying the difference between the two and this would increase the overall model accuracy.

Now let's look at the second issue, high volatility within a degree marker set. To do this, let's look at colors, specifically red, orange, and yellow. Most people can identify the difference between these three colors, however these same people may have issues identifying at what point a red becomes orange and at what point orange becomes yellow. In fact, depending on who you ask, these discrete color boundaries change; one person may identify one color as red, another will say it's orange. This is what happens when our data has too high volatility within each degree marker set. It becomes harder to put a discrete bound on our degree marker sets because a higher volatility means that we have to span a larger bound for a particular data field

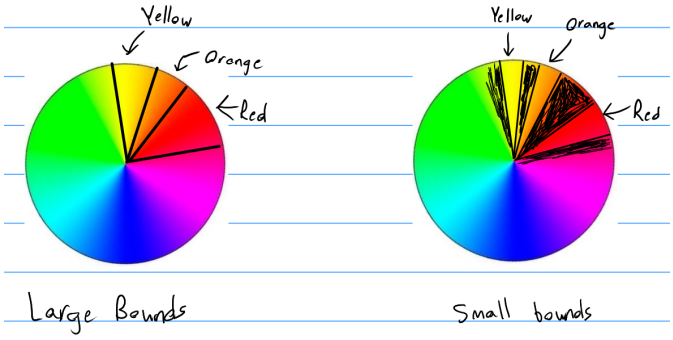

Figure 2: Color Wheel Example

Figure 2 is meant to show what happens when we combine lack of distinction between sets and high volatility within sets. Shown are two color wheels, one with lines drawn roughly (I am color blind so please be forgiving) between yellow, red, and orange shown on the left, and one with discrete ranges drawn for each of the three colors shown on the right. With the color wheel on the left, if we're near a boundary, there's a higher level of uncertainty as to what color we're actually looking at. The distinction between the colors is low and because of high volatility in each color data set, we have such large ranges for what is classified as red, yellow, and orange (in actuality, there is probably overlap between each bound meaning it can be either (red or orange)/(orange or yellow)). However on the right, we see distinct regions for the three colors. Because of high distinction within each set, it's clear to see where the boundaries exist for each color, and because of low volatility, we have very narrow bands for each color.

Now the color wheel on the right might upset some people because now we have portions of the wheel which are ignored, yet still represent a color. We begin to ask ourselves, "what happens when the DNN receives a point within these shaded regions?", and **hopefully** the model will produce a low confidence level for any one classification and thus we can ignore it. The idea is that our model will provide high confidence values in the non-shaded regions and low confidence in shaded regions, and if any one confidence level doesn't go above a certain threshold, we maintain our current prediction.

So in summary, our issue is that our model looks more like the color wheel on the left-hand side. In actuality, it probably looks like this:

Figure 3: Colors as Bars w/ Overlap

Where the range which defines one color overlaps very heavily with every other color.

3 The Algorithm

3.1 Introduction

Assuming you've read the background section, you've probably determined that this algorithm would give us something like the "small bounds" color wheel; and you'd be right. The algorithm I've thought up works in two main parts: filtering and classifying. Spoiler alert, filtering is the easy part. Filtering will seek to decrease overall volatility within the degree marker sets by removing fields within a degree marker set which have high variation. Classifying will involve selecting fields within a degree marker set which uniquely identify said degree marker set. It should be noted that these aren't discrete parts, filtering happens because of the classification.

3.2 Getting the Variance of Our Data

For clarification, a **degree marker set** is a set of data points for any given degree (i.e. 100 data points for 0°would be a degree marker set of size 100). A **data point** has 106 **fields** (53 for phase, 53 for magnitude). The first step is determining the volatility for every field in a degree marker set. Every data point has the same fields which means that we are determining the volatility of 106 fields for each degree marker set. The best way I can think to do this is through variance and standard deviation.

	0°				
	f1	f2	f3	f4	f5
p1	1	10	4	30	8
p2	2	20	3	15	12
p3	3	30	6	14	14
p4	2	20	8	20	13
p5	3	40	10	16	12
Mean	2.2	24	6.2	19	11.8
Variance	0.56	104	6.56	34.4	4.16
Stand Deviation	0.748331	10.19804	2.56125	5.865151	2.039608
	15°				
	f1	f2	f3	f4	f5
p1	6	15	9	50	9
p2	7	22	7	25	13
p3	8	33	10	34	15
p4	7	25	12	40	14
p5	8	44	15	36	13
Mean	7.2	27.8	10.6	37	12.8
Variance	0.56	98.96	7.44	66.4	4.16
Stand Deviation	0.748331	9.947864	2.727636	8.14862	2.039608

Table 1: Standard Deviation/Variance Example

In the above table is an example the first part of the algorithm. Here we have two degree

marker sets, 0°and 15°, also there are only 5 fields and 5 data points. While it may be tempting to strip out any field that has high variance, this may strip out useful information from the next step. Example: Assume at 0°, one field oscillates between 0 and 100, but at 15° it oscillates between 200 and 300. There exists a high variance but they're distinct.

3.3 Classification (a.k.a The Hard Problem)

This is where things get difficult, we need to pick fields that map to a certain degree marker set. To me, this screams a greedy, recursive algorithm. It's greedy because, for each degree marker set, we sort the fields by variation, smallest to largest. If we take a field, and give it a width equal to $4 * \sigma$, and we give it a set point at it's mean, then we can determine how unique a field is inside of a degree marker set in comparison to the other degree marker sets.

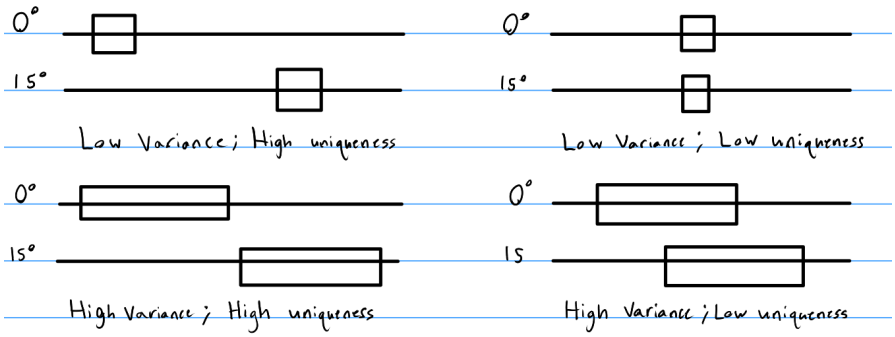

Figure 4: Variance vs Uniqueness

In the image above, there are 4 difference scenarios. The first is when there's low variance and the means, the set points, are far apart. For clarity sake, we'll say that the top line is field 1 for 0° and the bottom line is field 1 for 15°. If we state the the left and right bounds of the box are both two standard deviations away from the set point, we can say that for this field, if a point falls within the rectangle on the top line, that the door is at 0° and if it falls withing the rectangle on the bottom line, that the door is at 15°. The seconds scenario (low variance but similar means) illustrates that low variance doesn't always mean we can assume no overlap with other degree marker sets. However, I'd argue that it's more useful to check low variance fields first as these fields will have a better chance of being unique between data marker sets. Scenario 3 illustrates that we can have high variance but that overlap may not exist between two data marker sets. Finally, scenario 4 shows what I think is most common, high variance and lots of overlap. In our filtering step, we're going to remove/ignore fields which exhibit behaviors specified in scenario 2 and 4 as these are noisy. Ideally, we'll also want to cut down the number of scenario 3 fields we look at as this would increase the similarity of the data points within a degree marker set.

So if we go through and calculate the variance of each field within a degree marker set and we find that we have lots of scenario 1 fields, great, super easy. However I think the following scenario is more likely:

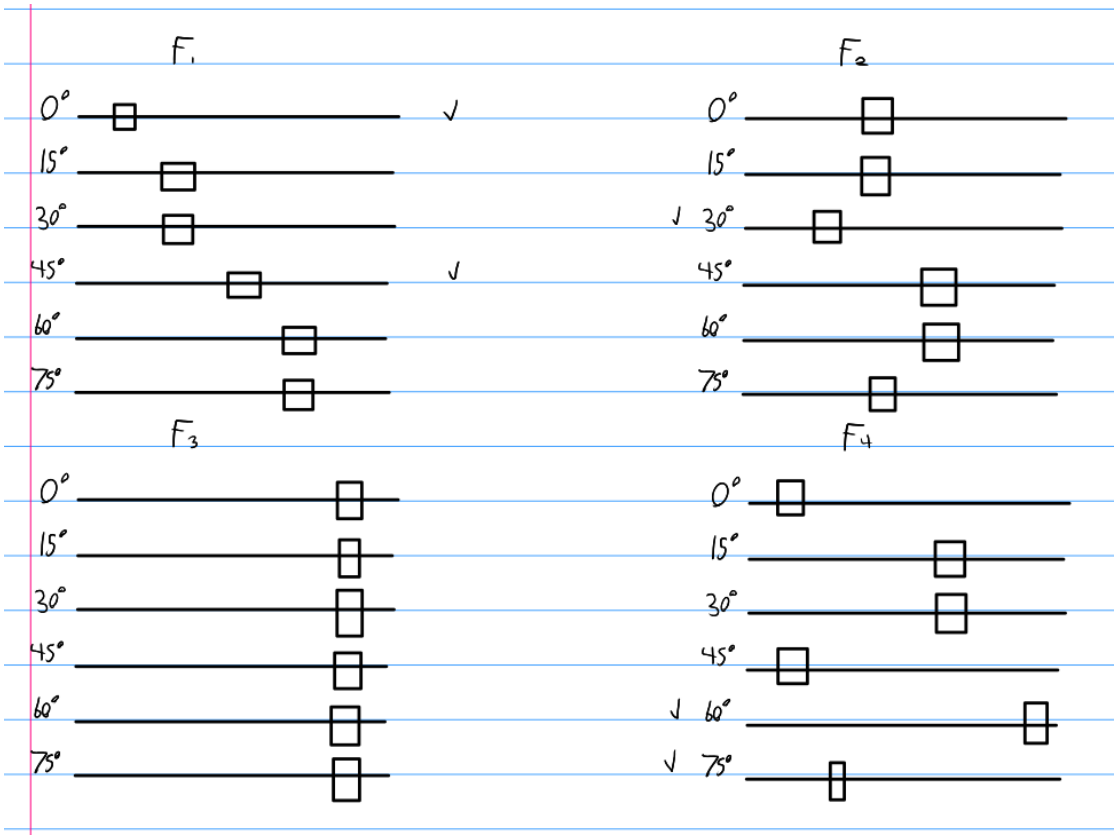

Figure 5: Classification Example

In figure 5, I've written out the sets for angles 0-75 for the first 4 fields. With this data set, we should be able to come up with fields and values to determine when the door is at any of the discrete angles. If we consider field 1, then if a data point comes in with a field value that falls within the 0° box or the 45° box, we can assume that it's either 0° or 45°. Similar things can be said about 30°, 60°, and 75°. But how about 15°?

This is where things get difficult. We can identify 15° if in field 1, it doesn't overlap with degree marker set 0's field 1 **AND** if in field 2, it doesn't overlap with degree marker set 30's field 2. There is an easy-ish approach to solving this problem though. Essentially if there is overlap for a n-given degree marker sets in a given field, we can look at all other fields for those n-given degree mark sets and if there is another field where overlap doesn't exist, then we can associate those two fields together. However, there's a caveat: fields are strongly related to one

another. This means that every place where field 1 has the value 30, there's a good chance the field 2 will have a very similar value each time. Because of this, we can only looks at points which meet the first fields criteria.

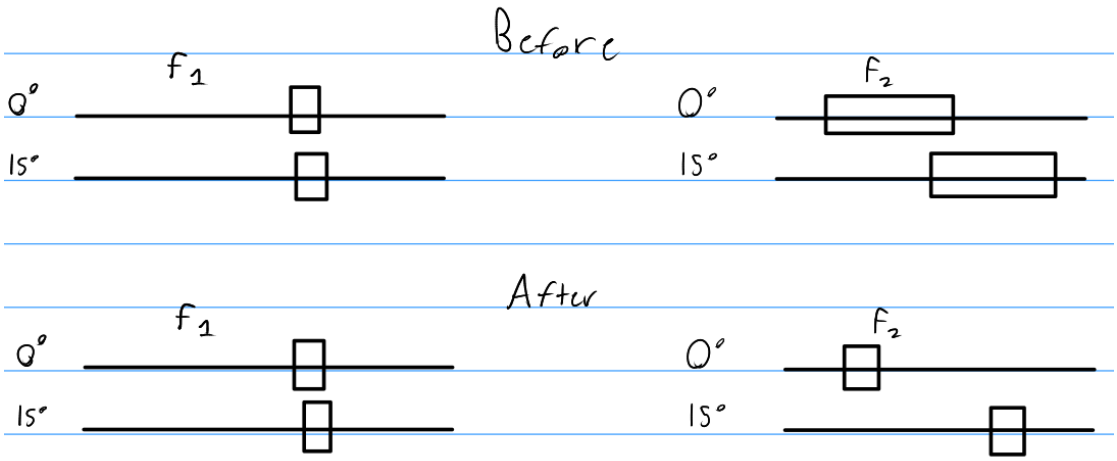

Figure 6: Recursive Search Example

So figure 6 should illustrate this a little better. Essentially, when we calculate the variation, standard deviation, and mean for field 1 and field 2, we see that there exists a high amount of overlap for both degree marker sets in both fields, however if we set the criteria that we allow for a overlap between the two sets in field one, then we should re-evaluate all the points which fall within that range. So for clarity sake, we look at the training data set (let's say it's a 100 points for each degree marker set) and we only look at the data points where field 1 falls within the bounds shown in the image. We do this for degree marker sets 0 and 15, and we recompute the mean, variance, and standard deviation for each of the fields using those data points. We may now find that there's significantly less variance and significantly more uniqueness between the two sets.

3.4 Conclusion/Summary

So to summarize the algorithm, we determine the variance of each field in a degree marker set, we look for unique entries for each degree marker set. If we can't find them, we select a field which we allow overlap and re-evaluate all the points which meet that field's criteria and hope for unique entries. Once we have the fields we want to look for, we ignore all other fields. By ignoring the other fields, we reduce overall noise (the Monalisa example) and increase the amount of unique data in the system. By making note of the fields that don't overlap, we now have a way to classify any given point (at this point I wonder if we even need a DNN to classify the data points).

4 Approaches to Classification

So one way to tackle the classification problem is by treating the entire problem as a multi-processor scheduling problem. We have n-processors where n is the number of data fields. Each processor has m tasks where m is the total number of degree marker sets. The task length is $4 * \sigma$ and the arrival time is the mean. This would look a lot like figure 5, however each figure would be a processor and each line would be a task.

The issue is that there doesn't exist (to my knowledge) a polynomial time algorithm for a multiprocessor system. I could take a modified myopic scheduling approach to the problem, however this still wouldn't be guaranteed to work.

5 Short Comings

So there are some short comings to this approach. Firstly, we are designing a classifier for the discrete angle values. However these are not small increments, these are 15° increments. We reach a catch-22 where small increments means it's harder to discern between states but larger increments means there's a wider margin of unknown. It could be the case that the field we use to determine 75° just so happens to meet the same criteria at 7.5°. Then when the door is moving from 0° to 15°, half way it'll say it's open to 75°. I don't know how to compensate for this in an efficient manner.

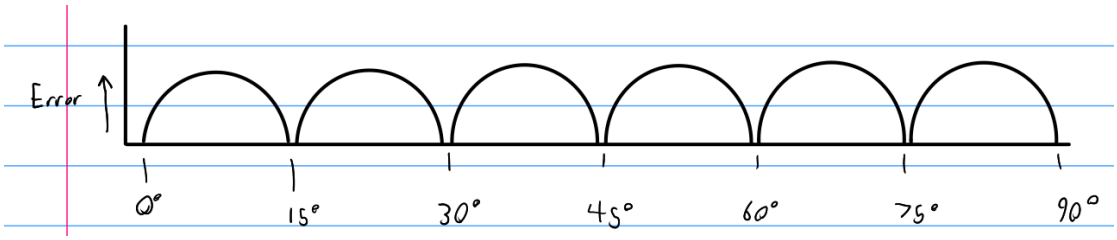

Figure 7: Algorithm Error

Shown in the figure above is what I'm expecting the error to look like. Very close to the degree marker, we're going to see very small error, however this is going to grow significantly as we move away from it. We could mitigate this to some degree by using a method mentioned earlier, essentially we only change our prediction when we calculate a very high confidence that we're at a certain degree marker.

This algorithm only works under the assumption that there are discrete fields (fields with no overlap with other fields) for each given degree marker set, or that we can recursively look for relations that provide locally discrete fields (in figure 6, the second line would be an instance of a locally discrete field). I don't know the data well enough to state whether this is true or not.

6 Conclusion

Assuming you've made it to this point, you should now understand that the "hard problem" is developing an algorithm to pick fields to look for each degree marker set. While I feel like the overall approach is straight forward and logical, it may prove difficult to translate to code. Considering I still don't know how to articulate the process in English, I feel like this may be a complex problem with no "good" answer.