

### Question 1: Download and Describe your choice of a real data set DS

This paper is my journey in trying to classify wattage used with explanatory variables that represent temperatures, humidity, wind speed, visibility, and dew point. This data set was provided by the University of California, Irvine Machine Learning Repository.

Originally the data set had a total of 29 attributes, described as follows:

Columns	Description
1	<b>Data and Time</b> (Year-Month-day hour:minute:Secod)
2-3	<b>Wattage</b> (Appliance's, lights)
4, 6, 8, 10, 12, 16, 18, 20	<b>Inside Temperatures in °C</b> (Kitchen, Living Room, Laundry, Office, Bathroom, Ironing Room, Teenager Room, Parents Room)
5, 7, 9, 11, 13, 17, 19, 21	<b>Inside % Humidity</b> (Kitchen, Living Room, Laundry, Office, Bathroom, Ironing Room, Teenager Room, Parents Room)
14, 22	<b>Outside Temperatures in °C</b> (North side of building, Chievres Weather Station)
15, 24	<b>Outside Percent Humidity</b> (North side of building, Chievres Weather Station)
23	<b>Pressure in mmHg</b>
25	<b>Wind Speed in m/s</b>
26	<b>Visibility in km</b>
27	<b>T dew point in °C</b>
28-29	<b>Random variables</b>

Table 1: Column Description of Original Dataset

The data had 19,735 observations that followed a timeline of 10-minute increments from January 11, 2016 to May 27, 2016. All of the attributes in this data set are continuous and don't really require any dummy variables.

My desire with this data set is to see how well I could classify wattage usage by groups of Low Wattage, Medium Wattage, and High Wattage, represented as 0, 1, and 2 respectively. Once I have it classified, I was hoping to see if there was some sort of relationship between temperature differential and wattage used.

Since all of my data was continuous, including my response variable, I needed to group my response variable into the 3 different classes. I did this by finding the quantiles of Wattage at 33.3% (53.33 Watts) and 66.6% (90 Watts) followed by distributing them to classes with the following method:

Class 0: Wattage < 33.3%

Class 1: 33.3% < Wattage < 66%

Class 2: 66% < Wattage

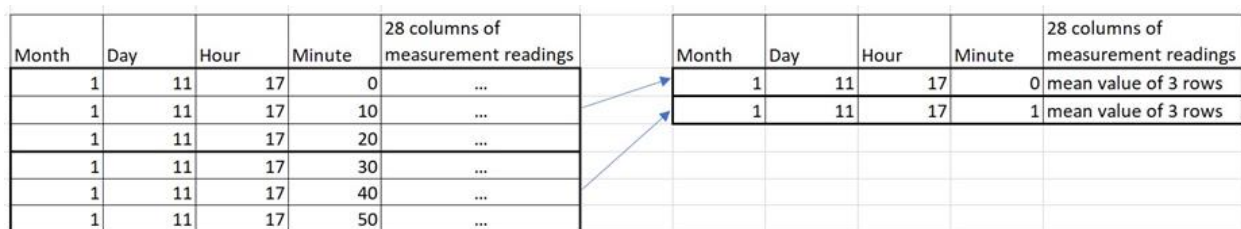
The next thing I had to do was to change the format of the date and time column and split that into 4 columns, Month, Day, Hour, Minutes. Since all of the data resides in one year, 2016, I went ahead and dropped this column. After splitting the data and time, I went ahead and deleted the two random variable terms added at the end of this data set. These were added to help filter the non-predictive

attributes and test regression formula. I also added three features to the data set based off of the already measured values. If you look at the table above you will see that 8 columns describe temperatures at different places inside and 2 columns describe temperatures outside. I took the average of all the inside temperatures for an average inside temperature and I took an average of the outside temperatures for an average of the outside temperatures. After this I got the difference of the temperature outside and inside and made that another column. The new table of column descriptions is displayed in the following table:

Columns	Description
1	<b>Class</b>
2	<b>Combined Wattage</b>
3,4,5	<b>Month, Day, Hour</b>
2-3	<b>Wattage</b> (Appliance's, lights)
6, 8, 10, 12, 14, 18, 20, 22, 24	<b>Inside Temperatures in °C</b> (Kitchen, Living Room, Laundry, Office, Bathroom, Ironing Room, Teenager Room, Parents Room, Average Inside)
7, 9, 11, 13, 15, 19, 21, 23	<b>Inside % Humidity</b> (Kitchen, Living Room, Laundry, Office, Bathroom, Ironing Room, Teenager Room, Parents Room)
16, 26, 27	<b>Outside Temperatures in °C</b> (North side of building, Average Outside Temperature, Chievres Weather Station)
17, 29	<b>Outside Percent Humidity</b> (North side of building, Chievres Weather Station)
25	<b>Temperature Difference</b> (avg temp outside – avg temp inside)
28	<b>Pressure in mmHg</b>
30	<b>Wind Speed in m/s</b>
31	<b>Visibility in km</b>
32	<b>T dew point in °C</b>

Table2: Column Description of Edited Dataset

After the feature engineering, I wanted to figure out a good way to reduce the amount of observations without just randomly selecting observations and potentially leaving a lot of data out. Since all of the variables are continuous and over time, I was able to take the mean over a larger time period. So instead of 10-minute increments, I reduced the data to 30-minute increments by the method displayed in the following image.



Month	Day	Hour	Minute	28 columns of measurement readings
1	11	17	0	...
1	11	17	10	...
1	11	17	20	...
1	11	17	30	...
1	11	17	40	...
1	11	17	50	...

Month	Day	Hour	Minute	28 columns of measurement readings
1	11	17	0	mean value of 3 rows
1	11	17	1	mean value of 3 rows

Fig 1: Observation Reduction Method

This simple consolidation of rows reduced the total number of observations down to 6579. Looking at the plots below, it is easy to see that it will be very difficult to group the classes by Wattage because that variable is very sporadic and kind of unpredictable. The second graph I sorted the temperature

difference and then plotted the wattage based off of this variable thinking there may be some trend. But as you can see, there is not.

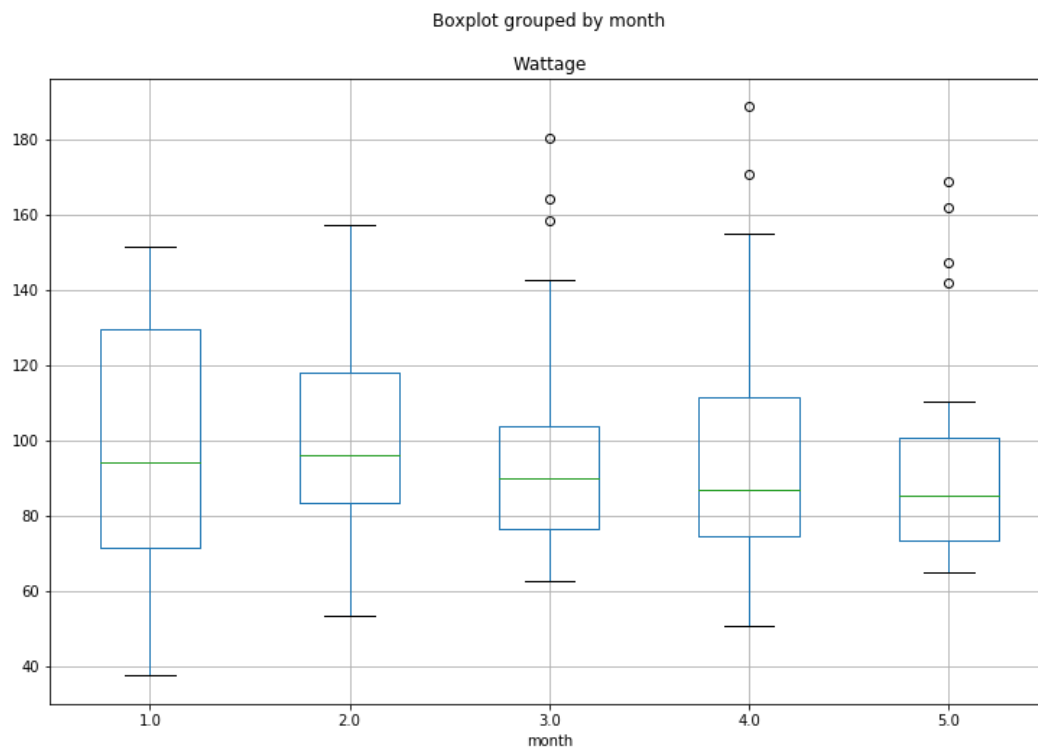


Fig 2: Box Plot of Wattage by Month

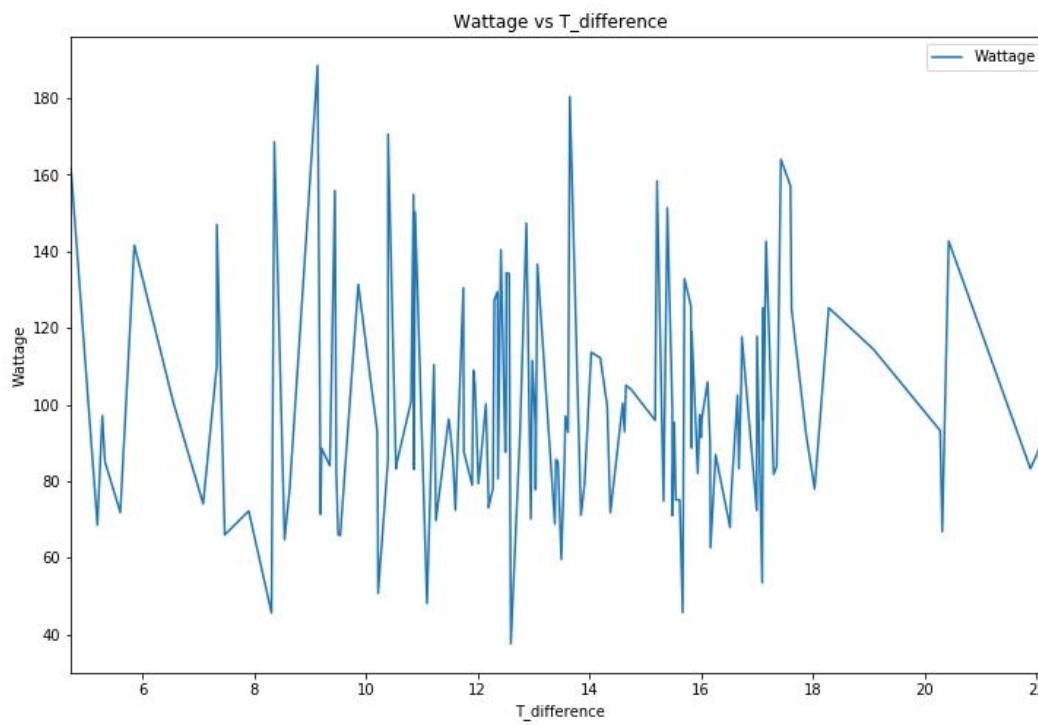


Fig 3: Wattage vs. rise in Temperature Difference

I also projected the x variables onto the first 3 eigenvectors found from the correlation matrix to help get an idea of the variance of the variables. The 3d plot below accounts for 74% of the variance in the x variables. As you can see, there seems to be some separation between the groups. I tried rotating the graph to see if there was any clear separation of the groups, but there were none that I could find. This makes me think there is some grouping going on, but not any defined and separate perimeters of the group.

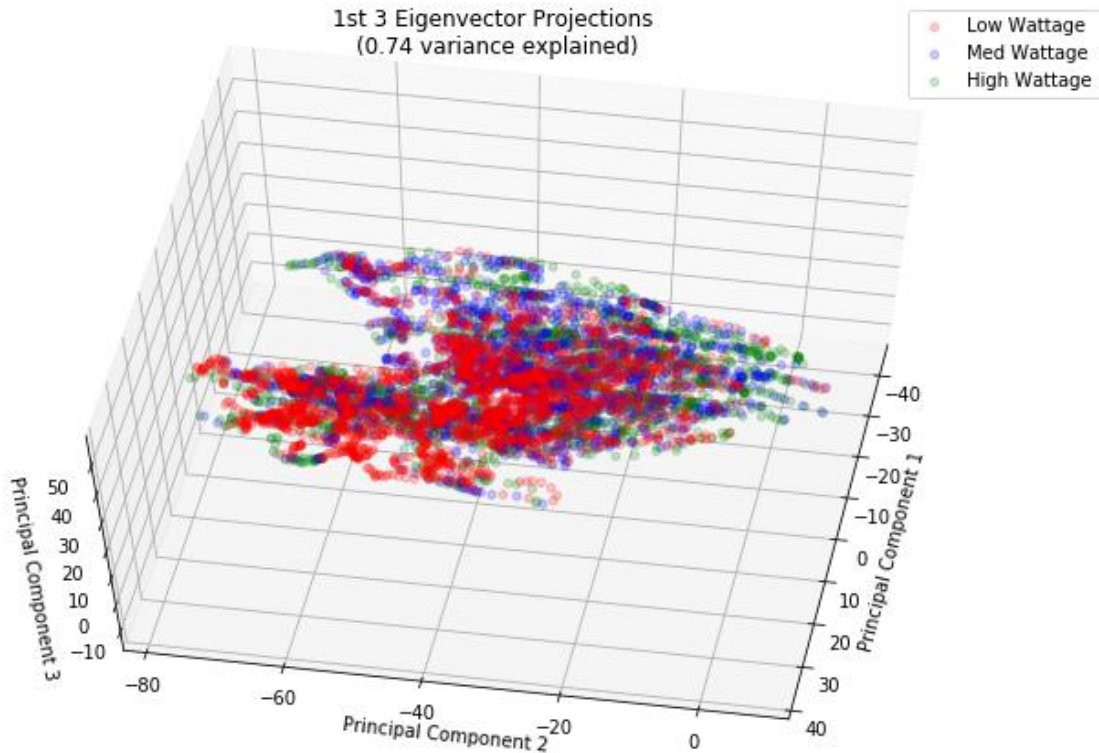


Fig 4: 3d plot of the groups

At this point I centered and rescaled the data with the following equations. For future use of scaling data for these SVM's I kept the mean and standard deviation of each column, displayed below also.

$$x = \frac{x_k - \bar{x}_i}{\sigma_{x_i}} \quad \forall \quad i = 1, 2, 3, 4 \quad ; \quad k = 1, \dots, n$$

Equation 1: Equation for standardizing a feature

$$\bar{x}_i = \frac{\sum_{k=1}^n x_{ki}}{n}, i = \text{columns}, n = \text{observations}$$

Equation 2: Equation for mean

$$\sigma_i = \sqrt{\frac{\sum_{k=1}^n (x_{ki} - \bar{x}_i)^2}{n}}, i = \text{columns}, n = \text{observations}$$

Equation 3: Equation for standard deviation

	Mean	SD		Mean	SD
month	3.102	1.339	T7	20.268	2.110
day	16.059	8.452	RH_7	35.389	5.113
hour	11.503	6.922	T8	22.029	1.956
T1	21.687	1.606	RH_8	42.937	5.222
RH_1	40.260	3.964	T9	19.486	2.015
T2	20.342	2.192	RH_9	41.553	4.149
RH_2	40.421	4.064	T_inside	20.816	1.813
T3	22.268	2.006	T_difference	13.163	4.514
RH_3	39.243	3.251	T_outside	7.663	5.668
T4	20.856	2.042	T_out	7.413	5.318
RH_4	39.028	4.340	Press_mm_hg	755.523	7.399
T5	19.592	1.843	RH_out	79.748	14.886
RH_5	50.949	8.887	Windspeed	4.040	2.447
T6	7.912	6.089	Visibility	38.330	11.664
RH_6	54.604	31.139	Tdewpoint	3.762	4.194

Table 4: Mean and Standard Deviation for each column

After standardizing we get a data set where the explanatory variables have a mean of 0 and variance of 1.

### TEST/TRAIN SPLIT

At this point, it is time to split the data for into train and test sets for parameter optimization and algorithm testing. I used the standard of 80%, 5262 observations, for train and 20%, 1317 observations, for test set. In doing this I made sure to split the inner classes equally by percentage, so 80% of each class individually makes up the 80% for the training set from the full data set. Since SVM is used for only separating between two classes, using SVM for 3 classes requires the building of multiple algorithms and looking at the story of all three algorithms to make a final decision in the end. The method in which I used was to tune each algorithm as 1 vs all. So, in the case of Low Wattage, or Class 0, I trained  $SVM_0$  with the two classes *Class 0 and Class 1  $\cup$  Class 2*. In my case, I did all the observations from the relative Class and SVM's as +1 and all the observation from the union of the other two Classes as -1. For instance, with  $SVM_0$ , Class 0's response variables were +1 and *Class 1  $\cup$  Class 2* response variables were -1. But, in doing it this way, I had a proportionality problem that skewed my training of the algorithms and required me to equalize the amount of cases that were +1 and -1. I will discuss this and show what it did later in the report, right now I am just explaining how I split up my data for the tuning and fitting of the algorithms. In the case of  $SVM_0$ 's original training set I had 1855 observations in +1 and 3407 observations not in +1. There are two ways I feel I could have of equalized proportionality; I could have cloned the +1 observations or I could reduce the -1 observations. The route I went was to reduce the -1 observations by half to get proportionality of about  $\frac{1}{2}$  of the observations in the training set in -1 and the other half in +1. One thing to note is that -1 is made up of two different Class's, I kept this in mind and took half of each class away to reduce the entire -1 by  $\frac{1}{2}$ . The left-over observations in the training set that I did not use, I put into the test set to increase the observations in the test set and potentially get a more reliable test score. This is for one class, this had to be done for each SVM and the following tables show you the splits I did.

Total			
(6579 observations)			
Train		Test	
Class 1	1855	Class 1	464
Class 2	1684	Class 2	422
Class 3	1723	Class 3	431
Total	5262	Total	1317

Table 5: Train and Test Set

SVM(0)		SVM(1)		SVM(2)	
Train		Train		Train	
+1	1855	+1	1684	+1	1723
-1	1703	-1	1788	-1	1769
Total	3558	Total	3472	Total	3492
Test		Test		Test	
+1	464	+1	422	+1	431
-1	2557	-1	2685	-1	2656
Total	3021	Total	3107	Total	3087

Table 6: Train and Test Set for each SVM

Then, after fitting the data and getting the corresponding test and train confusion matrices for each SVM, I compute the final classification with the final method using the full training set and the full test set. I did this because all of the training data had been used to train the algorithms, maybe not individually but if you look at all of the SVM's as a whole, which make up the final algorithm, they have seen every observation in the training set.

### Question 2/3: SVM classification by radial kernel (Tuning and Confusion Matrices)

The radial kernel is as follows:

$$k(x_k, x_i) = e^{-\gamma \|x_k - x_i\|^2} \quad \forall \quad k = 1, \dots, n_{\text{observations}} ; i = 1, \dots, n_{\text{support vectors}}$$

Equation 4: Radial Kernel Function

The method I went this time was to find an area to focus on by using grid search and then I used manual tuning so that I could look at all of the attributes that help make a decision in choosing an algorithm: Test Score, Train Score, and Ratio of Support Vectors. Looking at these things help show you accuracy with the scores and confidence in model with ratio of support vectors. The ratio of support vectors and score is found with the following equations:

$$\text{Ratio}_{\text{support vectors}} = \frac{S}{n_{\text{observations}}}$$

$$S = \text{number of Support vectors}, \quad n_{\text{observations}} = \text{observations}$$

Equation 5: Radial Kernel Function

$$Score = \frac{Predictions_{correct}}{n_{observations}}$$

Equation 6: Radial Kernel Function

## SVM<sub>0</sub>

To start I used the build in grid function that python has under their SciKit-Learn package. The parameters to optimize in this case was  $C$  and  $\gamma$ . Since I am doing this just to get an idea of where to start, I used the following values for the parameters:

$$\gamma: [.00001, .001, .1, 1]$$

$$C: [.00001, .1, 20, 60, 100]$$

Using this wide range, I was able to get an idea of where I wanted to focus my manual tuning, which really sped up the process of exploring the parameters and how it effects the algorithm accuracy and robustness. The actual running of the grid search function takes longer than the manual tune but being able to look at different combinations of the parameters allows me to focus in an area without having to just change one parameter at a time and seeing how this effect accuracy and robustness. The reason it takes so long is because it runs the algorithm for all of the different combinations in the list of parameters. In this case I have 5 parameters for  $C$  and 4 parameters for  $\gamma$ , you have 20 different combinations, shown below.

C	1E-05	1E-05	1E-05	1E-05	0.1	0.1	0.1	0.1	20	20	20	20	60	60	60	60	100	100	100	100
gamma	1E-05	0.001	0.1	1	1E-05	0.001	0.1	1	1E-05	0.001	0.1	1	1E-05	0.001	0.1	1	1E-05	0.001	0.1	1

Table 7: Combinations of Parameters

I also set the attribute cv to 10, meaning it does 10-fold cross validation. This means that for each combination, the grid search is randomly selecting 90% of the observations for a train set and 10% for a test set. It then runs this set of data through the algorithm with the parameters 10 times and takes the average test and train scores for each combination of parameters. The default value for the attribute cv in python is 3, which means it would do this split 3 times and run the algorithm 3 different times and get the average of those scores. But since we used a 10-fold cross-validation, the algorithm was actually running 200 different times and fit with different parameters 20 different times. This will significantly increase the amount of time it takes to run the algorithm, but by doing cross validation you help to reduce the factor of overfitting. Using the parameters shown above, I get the following results in graph form:

Test vs. Train by gamma's

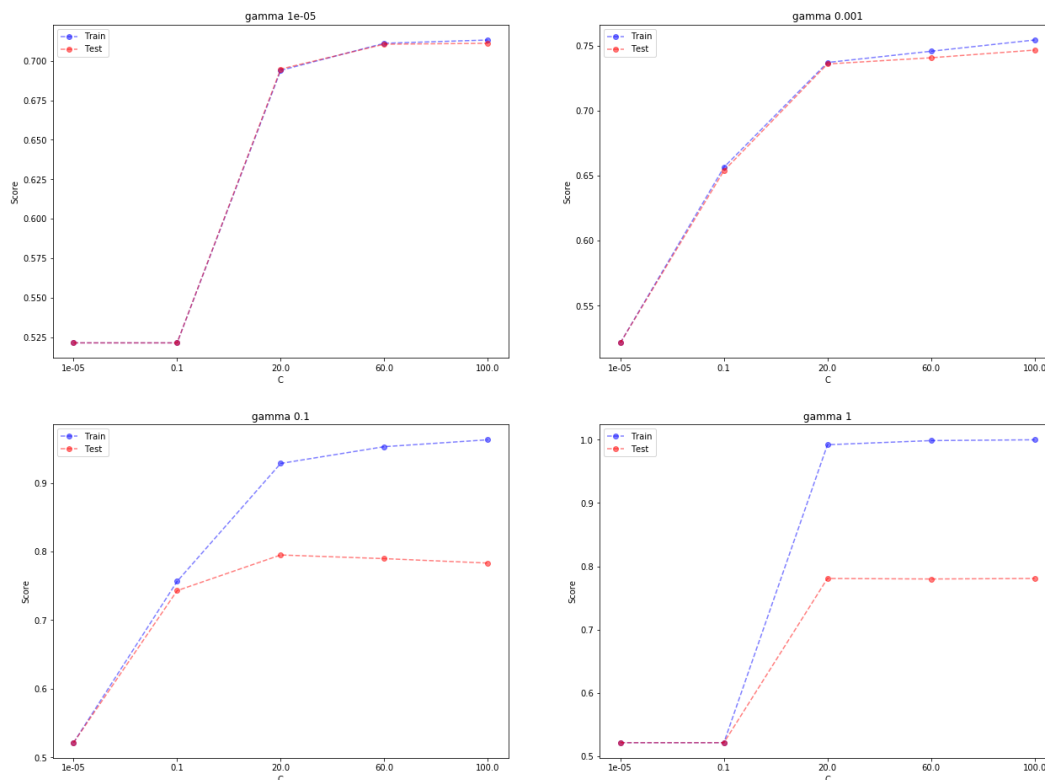


Fig 5: Grid Search Results for Parameters

Looking at the graphs above you can see that a cost of 20 has one of the better accuracies in all 4 different graphs. Then looking at gamma, I feel like the area to focus on falls between  $\gamma = 0.001$  and 0.1. The test score for a  $\gamma$  of 0.1 seems to be highest but there is also a pretty big difference between the test and train score, showing signs of low bias and high variance, meaning it may be slightly over fit. Looking at  $\gamma$  at 0.001, the score is slight lower but there is no difference between the test and train score making me feel more confident in the repeatability of the score. So, the parameters I choose to focus on in the manual tuning of parameters is a  $\gamma$  around 0.001 and a  $C$  of 20. The first parameter I tuned was the  $C$  factor with values of, [25, 30, 35, 40, 45, 50, 55], and I set  $\gamma$  to 0.01, splitting the two  $\gamma$ 's I like in the middle, and the results are displayed below. Looking at the graphs, you can see that changing the score changes the ratio of support vectors fairly significantly, but it did not have a major effect on score. You can see that a Cost of 50 seems to be the best, so I will progress to fine tuning  $\gamma$  with [.005, .0075, .01, .025, .05, .075, .1, .125], and fix the  $C$  to 50. This is graphed below the previous graph.



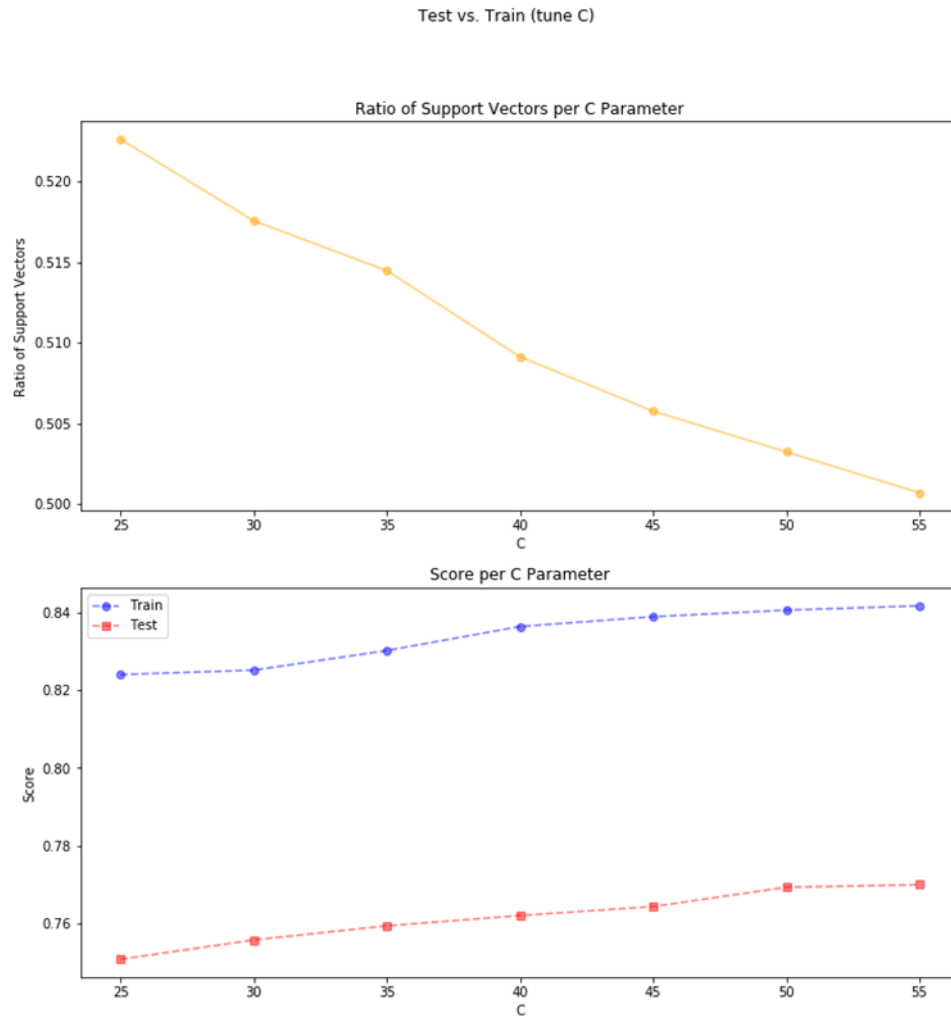


Fig 6: Ratio Support Vector and Train/Test Score relation for Manual Tune

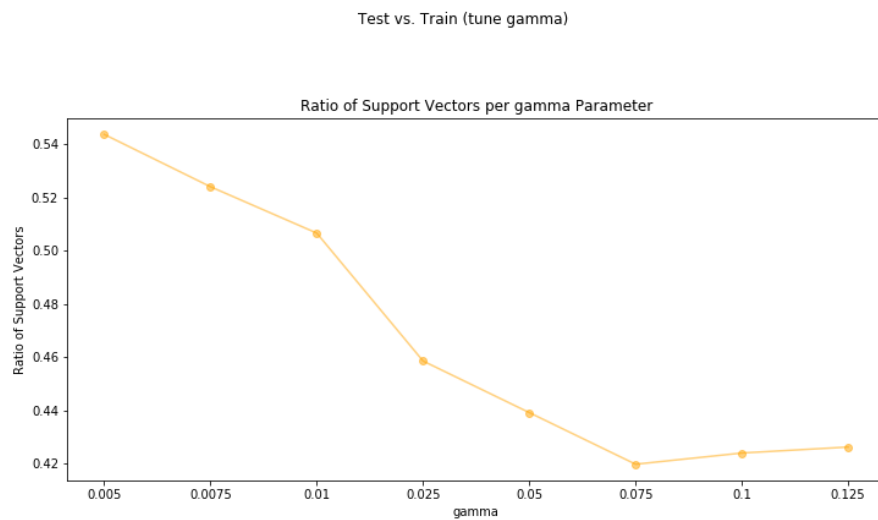


Fig 7: Ratio Support Vector for Manual Tune

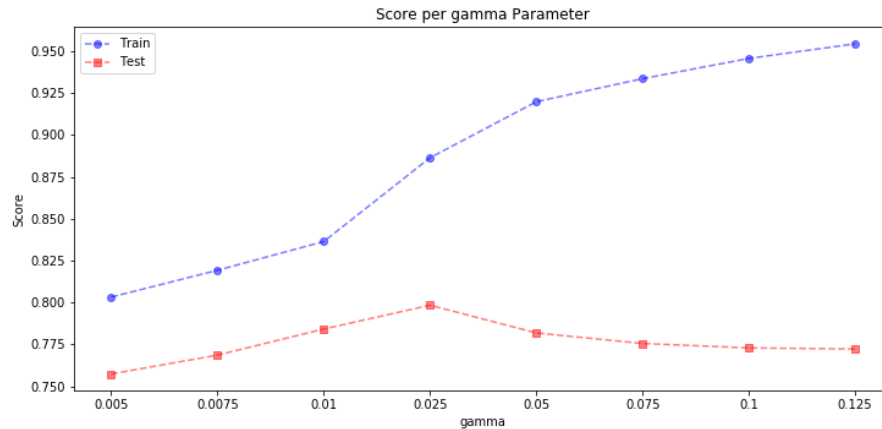


Fig 8: Train/Test Score relation for Manual Tune

Looking at the final graph for tuning, I chose the point where the test score breaks over and starts to spread away from the training score.

My conclusion from tuning  $SVM_0$  ended with parameters  $C = 50$  and  $\gamma = 0.025$ .

Now if you remember, we had to create a different training and test set for each SVM, so I will go ahead and show the confusion matrices for the relative training and test sets directly after the tuning of each function. The following table shows my results from running  $SVM_0$  on the training set and test set for  $SVM_0$ .

TRAIN					
		Prediction			
N(+1) = 1855		All Others (%)	Low Wattage (%)		
N(-1) = 1703				Sum (%)	
Actual	All Others	84.85	15.15	100.00	
	Low Wattage	7.87	92.13	100.00	
		Lower Limit (%)	Percent Correct (%)	Upper Limit (%)	Margin (%)
All Others		83.67	84.85	86.03	1.18
Low Wattage		91.24	92.13	93.01	0.88
Total		87.44	88.49	89.54	1.05
Ratio SV		0.458			

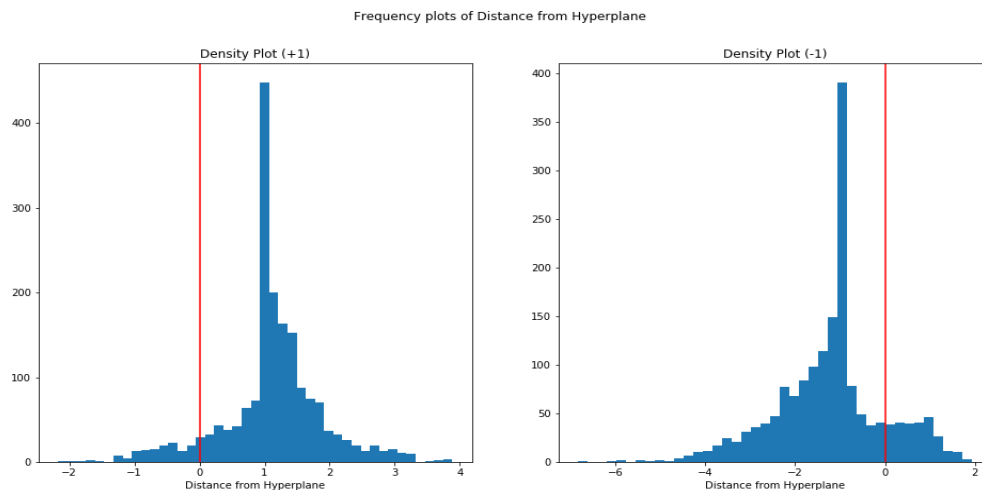
Table 8: Train Confusion Matrix and Limits for  $SVM_0$

Looking at the training confusion matrix, you can see that the algorithm was better at separating the +1's than the -1's. You can also see that the ratio of support vectors for  $SVM_0$  is 45.8% and to me that is not great. It makes me think that even in this higher dimension, a lot of the observations are not greatly separated.

TEST					
		Prediction			
N(+1) = 464 N(-1) = 2557		All Others (%)	Low Wattage (%)	Sum (%)	
Actual	All Others	79.19	20.81	100.00	
	Low Wattage	17.89	82.11	100.00	
		Lower Limit (%)	Percent Correct (%)	Upper Limit (%)	Margin (%)
All Others		77.75	79.19	80.64	1.45
Low Wattage		80.75	82.11	83.48	1.37
Total		79.24	80.65	82.06	1.41

Table 9: Test Confusion Matrix and Limits for  $SVM_0$ 

Looking at the results from the test set, you can see that it did a little worse than the training set but not so much so that it makes me feel there is over fitting going on. Over all I feel pretty confident about the chosen parameters for  $SVM_0$  and will now move on to  $SVM_1$ . One thing to help get an idea of reliability is to look at the following histograms.

Fig 9:  $SVM_0$  Histogram of Prediction's for  $Train_0$ 

This first graph is of the training set and it is pretty easy to see that the +1 class is grouped a little further than the separating hyperplane than the -1 class. The separating hyperplane is the 0 point on this graph, and the x-axis is the distance from the hyperplane. The graph of +1 is a graph of all the distances from separating hyperplane for the observations that are supposed to be in +1 class. That is why there are some values on the opposite side of the hyperplane, about 8%, which is shown in the training set confusion matrix. Looking at the -1 class you can see that a few more observations are incorrectly classified and more of the observations fall a little closer to the separating hyperplane. This gives me more confidence when the algorithm says it is in Class 1, but not as much when it says that is not in Class 1. The graph below shows the test sets histogram. You can see that both densities mean's fall closer the separating hyperplane. You may also notice that the -1 seems to have a lot more bins than the +1. This is

because I chose bins not as a fixed value for all, but as a factor of the number observations in the set. I simply chose the  $\sqrt{\text{observations}}$  to get this, there are much better methods in finding this but I have not had time to perfect my method in this instance and this does a better job than just fixing the number of bins.

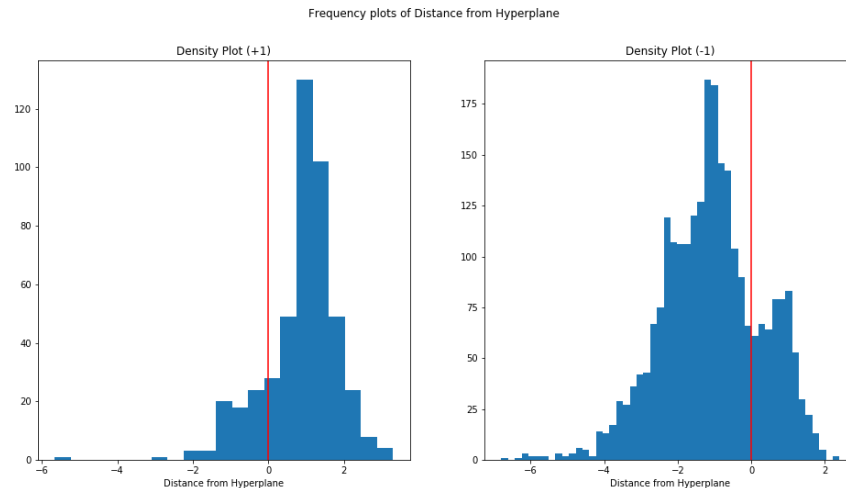


Fig 10: SVM<sub>0</sub> Histogram of Prediction's for Test<sub>0</sub>

## SVM<sub>1</sub>

With SVM<sub>1</sub>, I did my grid search with the following parameters.

$$\gamma: [.00001, .001, .1, 1]$$

$$C: [.00001, .1, 20, 60, 100]$$

Looking at the results of the following graph, it is looking again like a Cost around 20 and  $\gamma$  between 0.001 and 0.1 is where we want to be. Another thing I noticed when looking at the results from this grid search was how poorly it scored compared to SVM<sub>0</sub>. This makes sense to me because of how I formed the data set. Class 1 is the medium wattage, which mean it has low wattage on either side of it. SVM<sub>0</sub> was low wattage verse medium and high wattage. In that case, high wattage is not a direct neighbor to low wattage and the errors mostly come from medium wattage. In SVM<sub>1</sub> case, both classes it is separating from are direct neighbors, making me think that it will receive more errors than SVM<sub>0</sub> and SVM<sub>2</sub>. This same idea also makes me think that this model will be built with a lot more support vectors than the other two models. This will be seen in the manual tuning of parameters.

Test vs. Train by gamma's

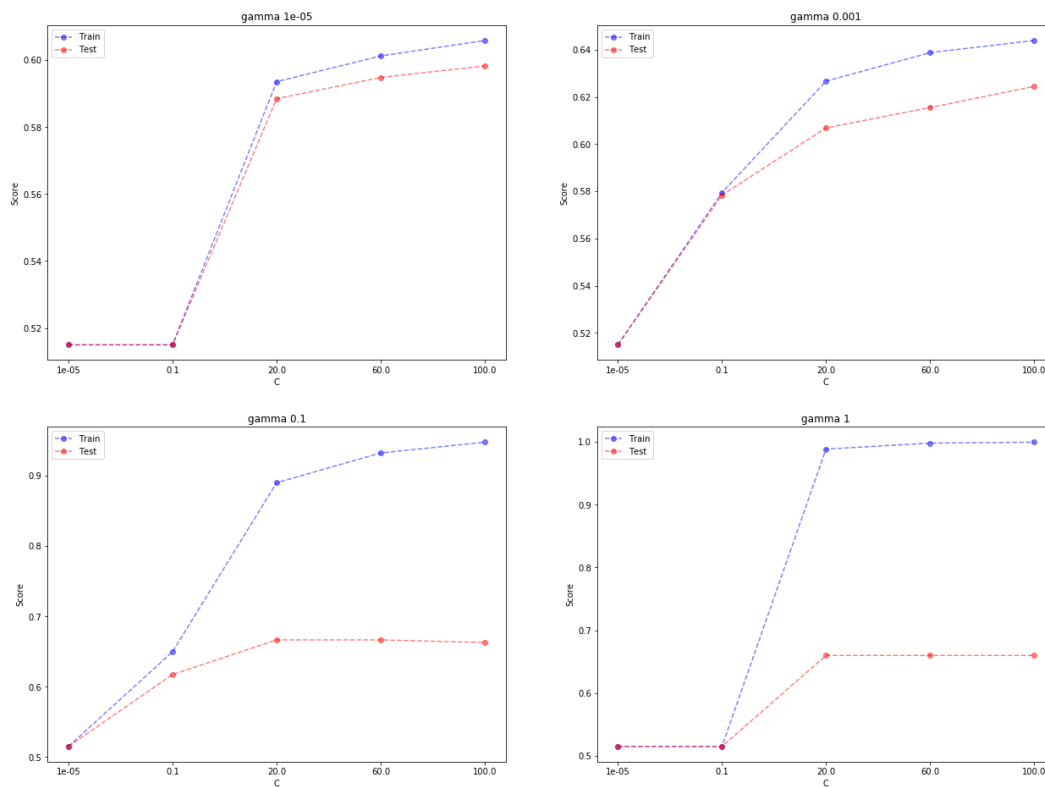


Fig 11: Grid Search Results for Parameters

Focusing on a gamma value of 0.1 and C values of [10, 15, 17.5, 20, 22.5, 25, 30, 35] we get the following graphs. Again, you can see cost does not make much of a difference to the score of test and train, but does have an effect on ratio of support vectors. After looking at these two graphs, I went with a C of 22.5. I chose this because I felt the scores for test and train separation was just growing after this point and test never really got any better. I didn't feel that the ratio of support vectors really changed between 20 and 25.

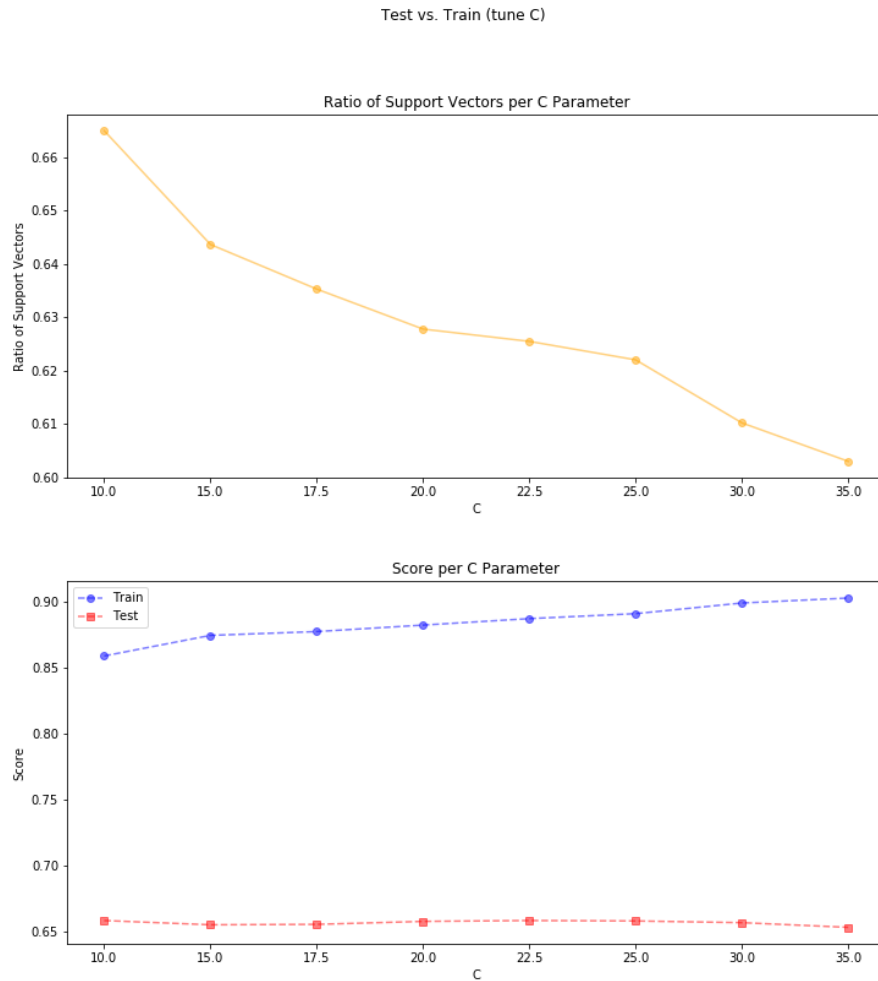


Fig 12: Ratio Support Vector and Train/Test Score relation for Manual Tune

Moving forward with a cost of 22.5, I now looked at how the SVM<sub>1</sub> acted with  $\gamma$  as [.01, .025, .05, .075, .1, .125, .150, .175] and got the following graphs.

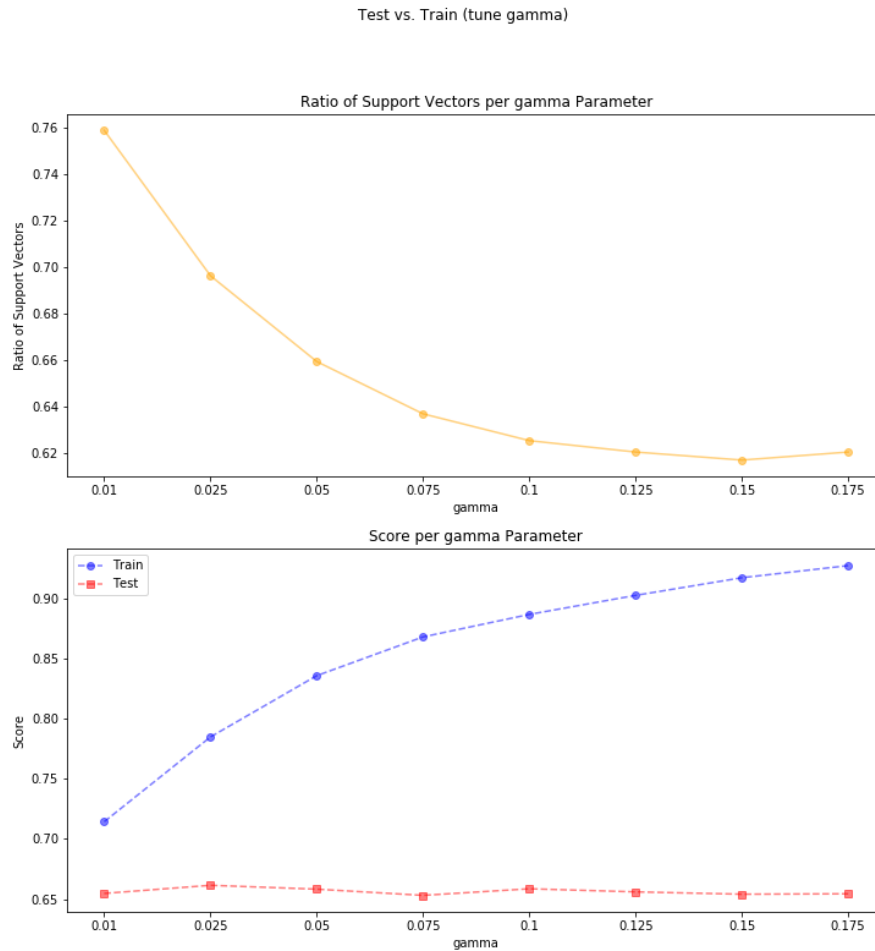


Fig 13: Ratio Support Vector and Train/Test Score relation for Manual Tune

The different  $\gamma$ 's really effected the ratio of support vectors until it got to 0.1, they really effected the training set score, and they did not really have an effect on the test score. Looking at these graphs I went with  $\gamma = 0.05$  because the combination of test-train difference and ratio of support vectors seemed like a good compromise. I tried to choose the highest test score with a relatively close train score and relatively low ratio of support vectors.

My conclusion from tuning SVM<sub>1</sub> ended with parameters  $C = 22.5$  and  $\gamma = 0.05$ .

The following table shows my results from running SVM<sub>1</sub> on the training set and test set for SVM<sub>1</sub>.

TRAIN					
		Prediction			
Actual	N(+1) = 1684	All Others (%)	Med Wattage (%)	Sum (%)	
	N(-1) = 1788				
	All Others	82.66	17.34	100.00	
	Med Wattage	14.49	85.51	100.00	
		Lower Limit (%)	Percent Correct (%)	Upper Limit (%)	Margin (%)
All Others		81.40	82.66	83.92	1.26
Med Wattage		84.34	85.51	86.68	1.17
Total		82.87	84.09	85.30	1.22
Ratio SV		0.653			

Table 10: Train Confusion Matrix and Limits for SVM<sub>1</sub>

You can see from the training set results that the overall score is  $84.09 \pm 1.22\%$  with a slightly better classification of Med Wattage (+1) than not all others (-1). This is not a whole lot worse than SVM<sub>0</sub>'s results, but the ratio of support vectors took quite a jump. This doesn't help my confidence in this model.

TEST					
		Prediction			
Actual	N(+1) = 422	All Others (%)	Med Wattage (%)	Sum (%)	
	N(-1) = 2685				
	Med Wattage	65.07	34.93	100.00	
	All Others	31.28	68.72	100.00	
		Lower Limit (%)	Percent Correct (%)	Upper Limit (%)	Margin (%)
All Others		63.39	65.07	66.74	1.68
Med Wattage		67.09	68.72	70.35	1.63
Total		65.24	66.89	68.55	1.65

Table 11: Test Confusion Matrix and Limits for SVM<sub>0</sub>

Looking at the test set results, it further proves that we should not be very confident in this model. That is a pretty decent spread between a test score of  $66.89 \pm 1.65$  and *train score*  $84.09 \pm 1.22\%$ . The combination of this spread and the high ratio of support vectors make me think this model has pretty high variance. Looking at the histograms should support these thoughts and are displayed below.



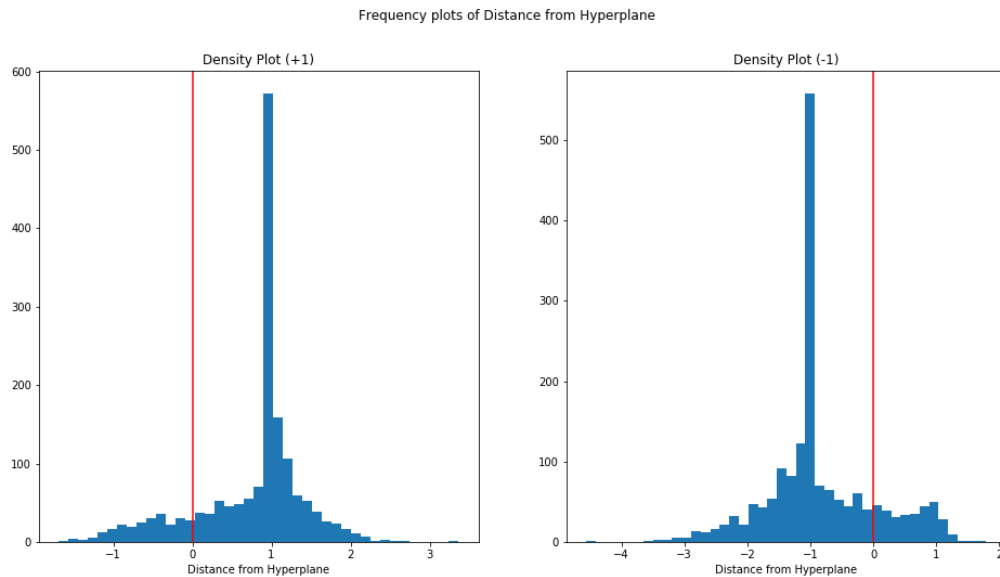


Fig 14: SVM<sub>0</sub> Histogram of Prediction's for Train<sub>0</sub>

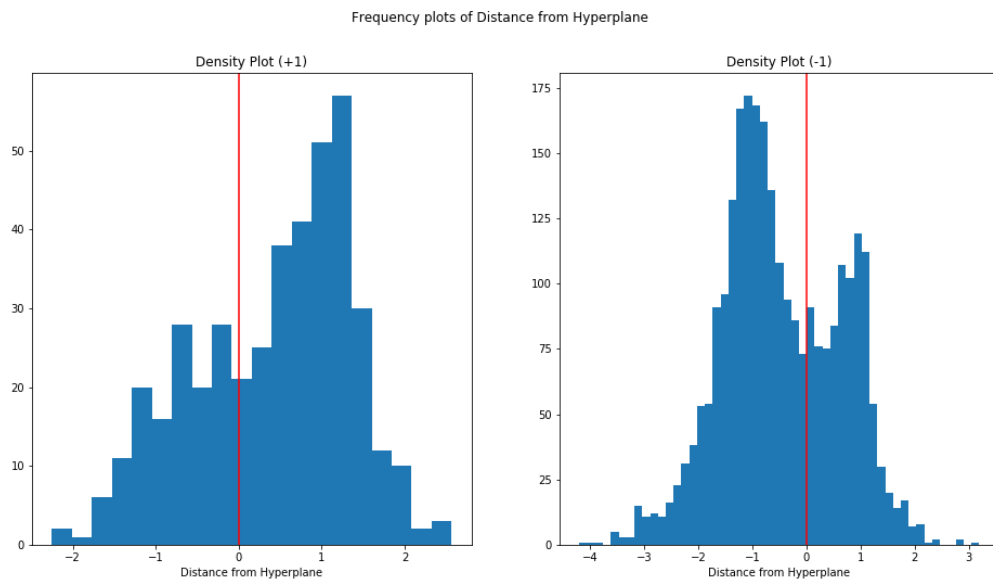


Fig 15: SVM<sub>0</sub> Histogram of Prediction's tor Test<sub>0</sub>

Looking at the histogram for the training set, it appears to show decent separation, although if you look at the scale of the x-axis, it is slightly less that shown on the previous SVM. Looking at the test histograms you can really see that there is another small distribution that seems to fall on the wrong side of the histogram. This was also seen on the test set of the last SVM for the -1 class. Maybe these are related and it could be worth a deeper dive, not in this report though.

## SVM<sub>2</sub>

With SVM<sub>2</sub>, I did my grid search with the following parameters.

$\gamma$ : [.00001,.001,.1,1]

$C$ : [.00001,.1,20,60,100]

Test vs. Train by gamma's

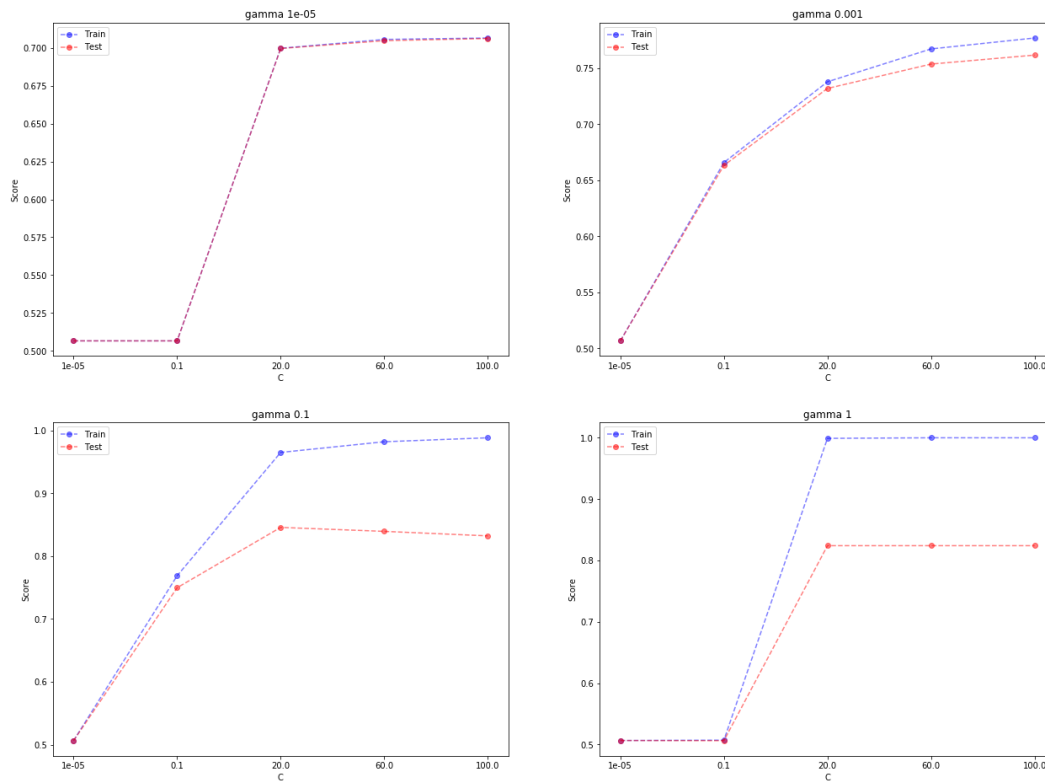


Fig 16: Grid Search Results for Parameters

Looking at the graphs above, I ended with the same areas to focus on and around:  $C = 20$ , and  $\gamma$  between 0.001 and 0.1. Using a gamma of 0.1, I looked at how the  $C$  values of [10, 15, 17.5, 20, 22.5, 25, 30, 35] would affect the model's output. With these parameters I got the following results.

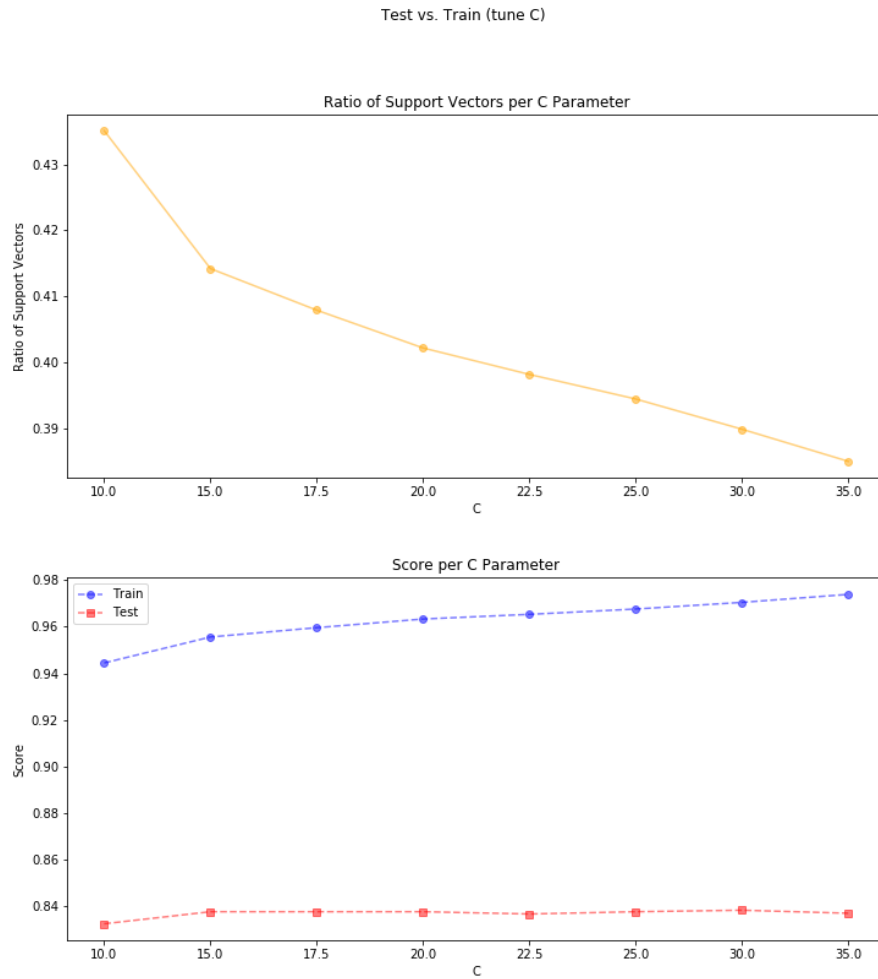


Fig 17: Ratio Support Vector and Train/Test Score relation for Manual Tune

Fixing the cost at 20 and using [.01, .025, .05, .075, .1, .125, .150, .175] for  $\gamma$  I got the following graph.

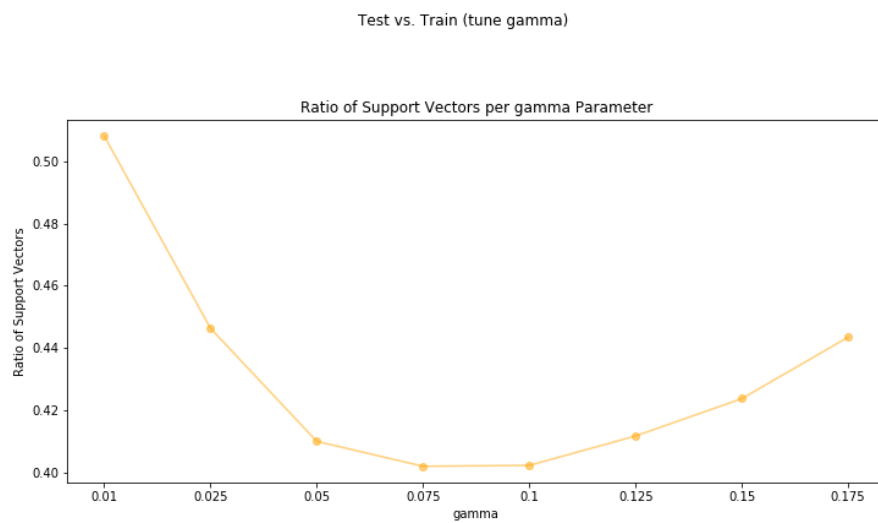


Fig 18: Ratio Support Vector for Manual Tune

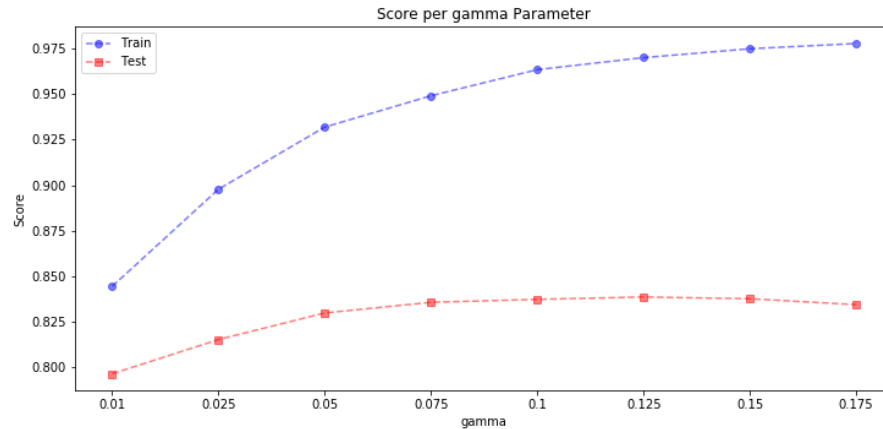


Fig 19: Train/Test Score relation for Manual Tune

Looking at this final graph I chose  $\gamma$  of 0.75 because it had the lowest ratio of support vectors and still an ok spread between test and train scores.

My conclusion from tuning SVM<sub>2</sub> ended with parameters  $C = 20$  and  $\gamma = 0.075$ .

The following table shows my results from running SVM<sub>2</sub> on the training set and test set for SVM<sub>2</sub>.

TRAIN					
		Prediction			
	N(+1) = 1723	All Others (%)	High Wattage (%)		
	N(-1) = 1769			Sum (%)	
Actual	All Others	94.12	5.88	100.00	
	High Wattage	4.29	95.71	100.00	
		Lower Limit (%)	Percent Correct (%)	Upper Limit (%)	Margin (%)
	All Others	93.34	94.12	94.90	0.78
	High Wattage	95.03	95.71	96.38	0.67
	Total	94.18	94.91	95.64	0.73
	Ratio SV	0.389			

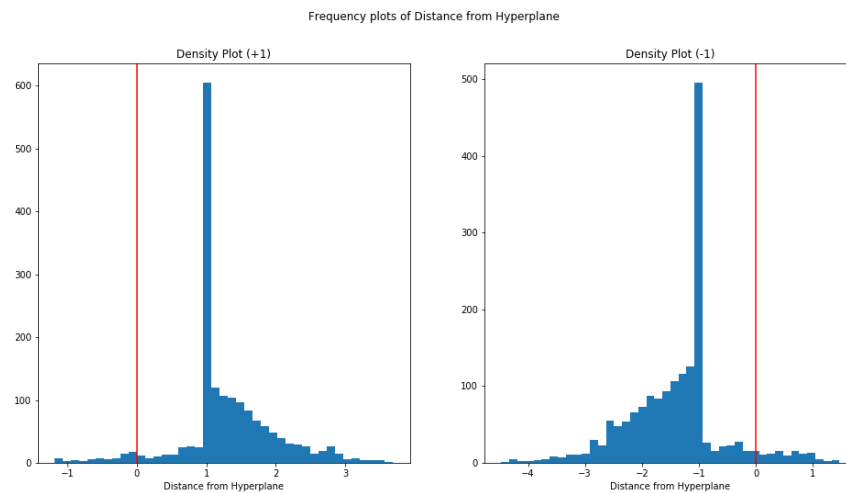
Table 12: Train Confusion Matrix and Limits for SVM<sub>2</sub>

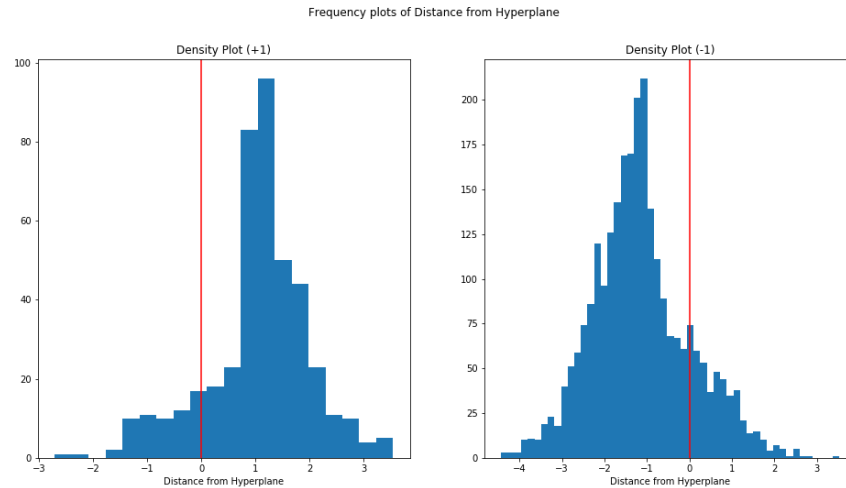
Looking at the training set's results you can see that this model does a better job of separating than the other two models. The train scores are higher and the ratio of support vectors is relatively low. I would still like to see this number lower, maybe between 10 and 20%, but I am not sure that will happen with this data set because these groups are pretty close to each other, as seen in the 3d plot earlier.

TEST					
		Prediction			
Actual	N(+1) = 431	All Others (%)	High Wattage (%)	Sum (%)	
	N(-1) = 2656				
	All Others	84.07	15.93	100.00	
	High Wattage	12.99	87.01	100.00	
		Lower Limit (%)	Percent Correct (%)	Upper Limit (%)	Margin (%)
All Others		82.78	84.07	85.36	1.29
High Wattage		85.82	87.01	88.19	1.19
Total		84.30	85.54	86.78	1.24

Table 13: Test Confusion Matrix and Limits for SVM<sub>2</sub>

The test set scores show that this model does a decent job separating data that has not been seen by the algorithm as well. Looking at the histograms we can see that there is pretty good separation and can be fairly confident in this model. The distance doesn't get much further than a distance of 1 from the hyperplane, but the spread on the test sets density graph is a bit tighter, which makes me feel more confident.

Fig 20: SVM<sub>2</sub> Histogram of Prediction's for Train<sub>2</sub>

Fig 21: SVM<sub>2</sub> Histogram of Prediction's for Test<sub>2</sub>

### Tuning and Training without Proportionality

Before I ran through this process with proportional classes in the training sets for each SVM, I ran through it using the entire training set. The Accuracies were not terrible but when I went to look at the matrices, I noticed that for Class 1, Medium Wattage, it classified everything as -1. When I was getting scores from the tuning process, I was getting about 67%. As I went on, I noticed that I was getting a score of 67% because 67% of the data was in class -1, not because it was actually making a prediction and it was 67% accurate. Refer to the confusion matrix below.

		Prediction		
	N(+1) = 1684	All Others (%)	Med Wattage (%)	
	N(-1) = 3578			Sum (%)
Actual	All Others	100.00	0.00	100.00
	Med Wattage	100.00	0.00	100.00

Table 14: Train Confusion Matrix without Proportionality

I then looked at the histogram and something even more interesting is that it classified everything as a distance of -1 from the hyperplane. I am not 100% as to why this is the case. But as I started to play with the proportionality of the training sets, I could see a histogram start to shift relative to the hyperplane. From my exploration with different proportions it seemed that 50% +1 and 50% -1 seemed to be the best, so this is how I proceeded.

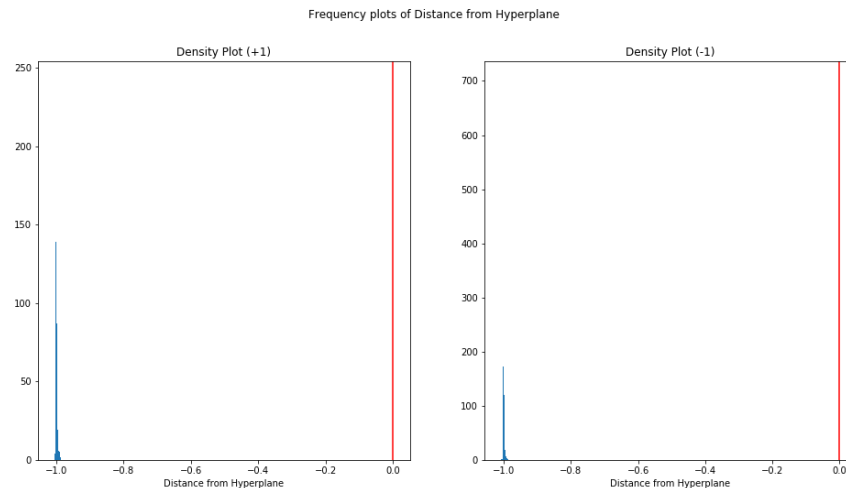


Fig 22: Histogram of Prediction's without taking into account Proportionality

**Question 4: Combine all SVM's to classify all cases (terminal classification and 3x3 confusion matrices)**

The method I used to classify each prediction is by taking advantage of the histograms I already have created for each SVM by finding the CDF for correctly classified +1's. If you have a histogram, finding the CDF is done pretty easily with the python scipy.stats package. By getting the CDF of the correctly classified +1's for each SVM, I can get a probability based on the distance from hyperplane of that observation if it is a +1 prediction. So, for every single observation, I will run its distance through the  $CDF_0$  for  $SVM_0$ , the  $CDF_1$  for  $SVM_1$ , and the  $CDF_2$  for  $SVM_2$  and I will get a probability for each prediction. If the distance is negative, it will give a probability of 0. If the distance is positive, it will give a probability that grows with the distance from the hyperplane. Once I have the probability of correct prediction for each SVM, I classified that prediction based on the highest probability given by the CDF's. I did this because that means it is the furthest from the hyperplane and I am really confident in that choice. I will try to help visualize an example under  $SVM_0$ . One thing to note is that I used the CDF created by the relative training set's PMF for each SVM's +1 classification. In real life, I would not have the correct response variable because that is what we are estimating. Since I would not be able to create any sort of distribution based off of the data coming in, I assign a class based off of the fitted SVM's CDF's from the training sets.

 **$SVM_0$ :**

With the training set I used for  $SVM_0$ , I get the following distribution for correctly assigned +1's. Looking at this distribution, it would appear that my confidence in a prediction that is a distance of +1 or more from the hyperplane should sharply increase because there are so many observations that are this distance away. If you look at the  $CDF_0$  below the distribution, you can see how it follows this thought.

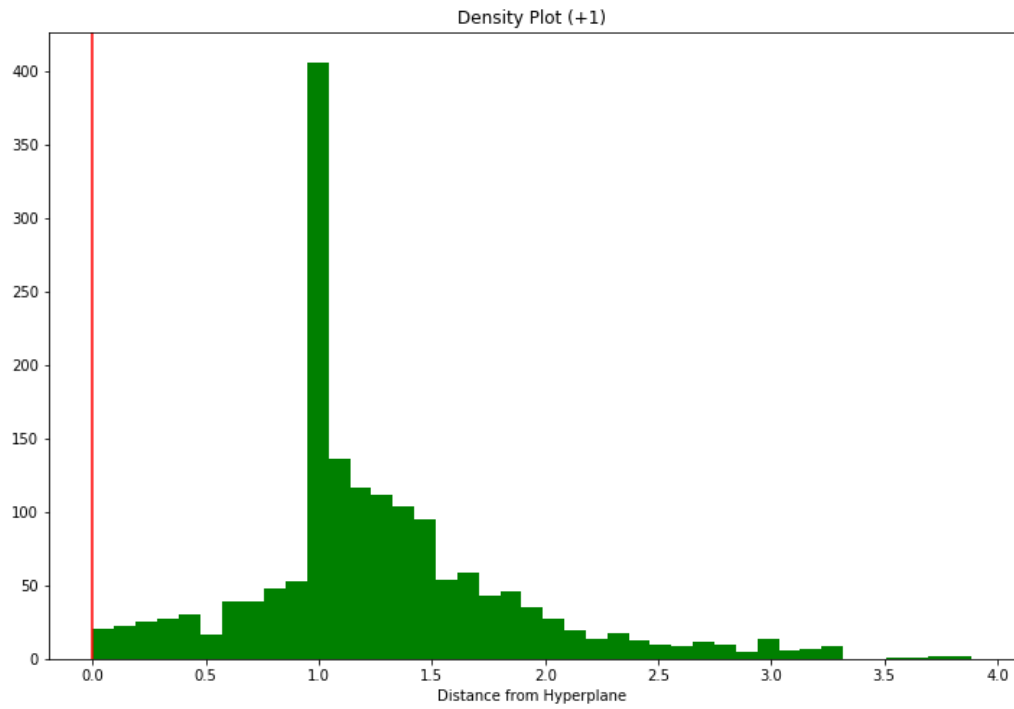


Fig 23: SVM<sub>0</sub> Histogram of Correct +1 Prediction's for Train<sub>0</sub>

The CDF below was built off of the distribution above.

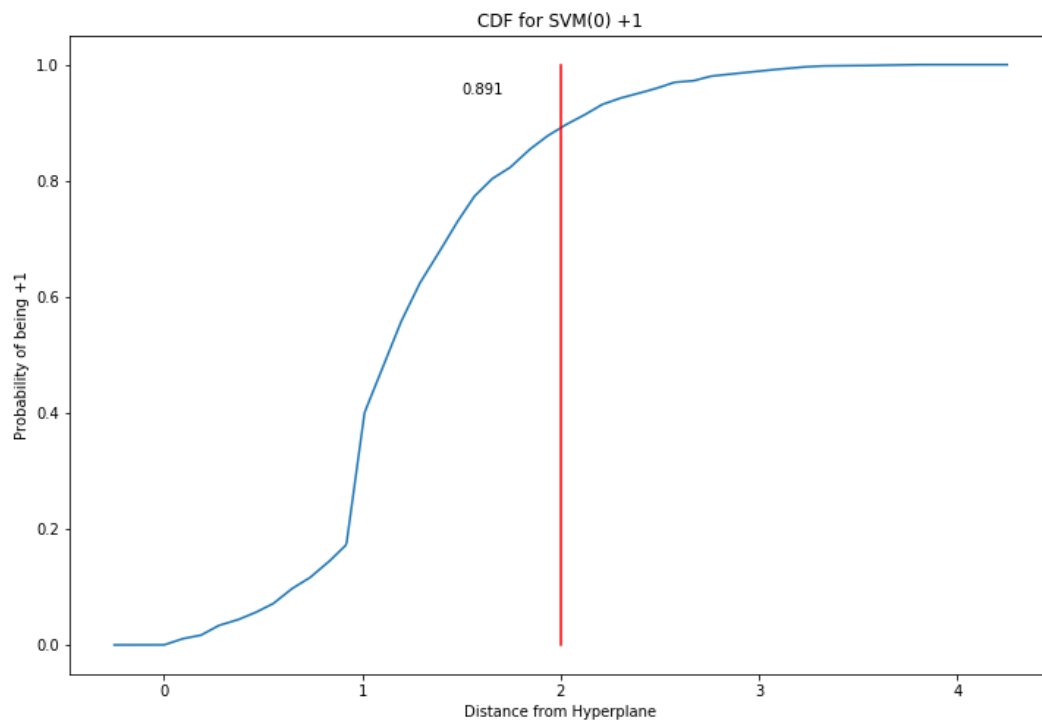


Fig 24: SVM<sub>0</sub> CDF of Correct +1 Prediction's for Train<sub>0</sub>



If we take an observation that is a distance of +2 away from the hyperplane, we will get a probability of 89.1% likely to be in class 0, or Low Wattage. If we run the entire training set's distance from hyperplane, we will get a probability based on the distance from the hyperplane, as shown in the first 10 cases of the training set.

	0	1	2	3	4	5	6	7	8	9
<b>SVM0</b>	0.160	0.448	0.448	0.000	0.000	0.000	0.000	0.000	0.000	0.013

Table 15: SVM<sub>0</sub> Confidence in 1<sup>st</sup> Nine Observations of Train Set

### SVM 1:

Plotting the histogram of correctly classified +1 responses from SVM<sub>1</sub> looks like the histogram below. This histogram is a PMF that creates a CDF<sub>1</sub>, which is used to give a probability based off of the distance from the hyperplane.

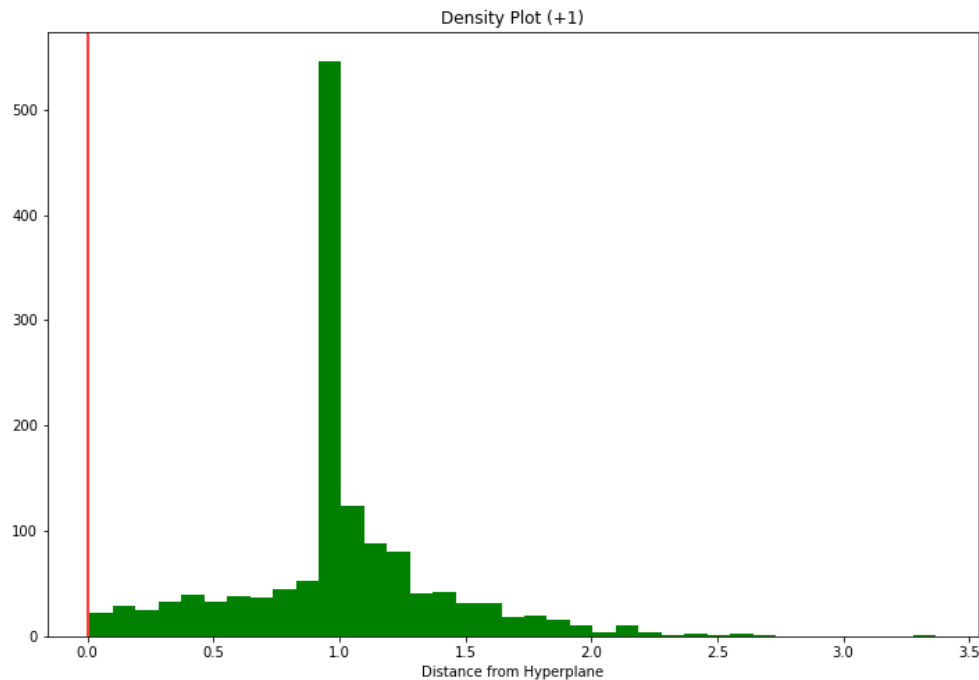


Fig 25: SVM<sub>1</sub> Histogram of Correct +1 Prediction's for Train<sub>1</sub>

Using the CDF<sub>1</sub> below for the first 10 cases of the training set, as I did earlier, you get the following probabilities. You can see that the very first observation in the training set has a 54.7% chance of being in Class 1, or Med Wattage. The ones that are 0 are said to not be in Class 1 by SVM<sub>1</sub>.

	0	1	2	3	4	5	6	7	8	9
<b>SVM1</b>	0.547	0.134	0.056	0.119	0.059	0.000	0.000	0.000	0.000	0.054

Table 16: SVM<sub>1</sub> Confidence in 1<sup>st</sup> Nine Observations of Train Set

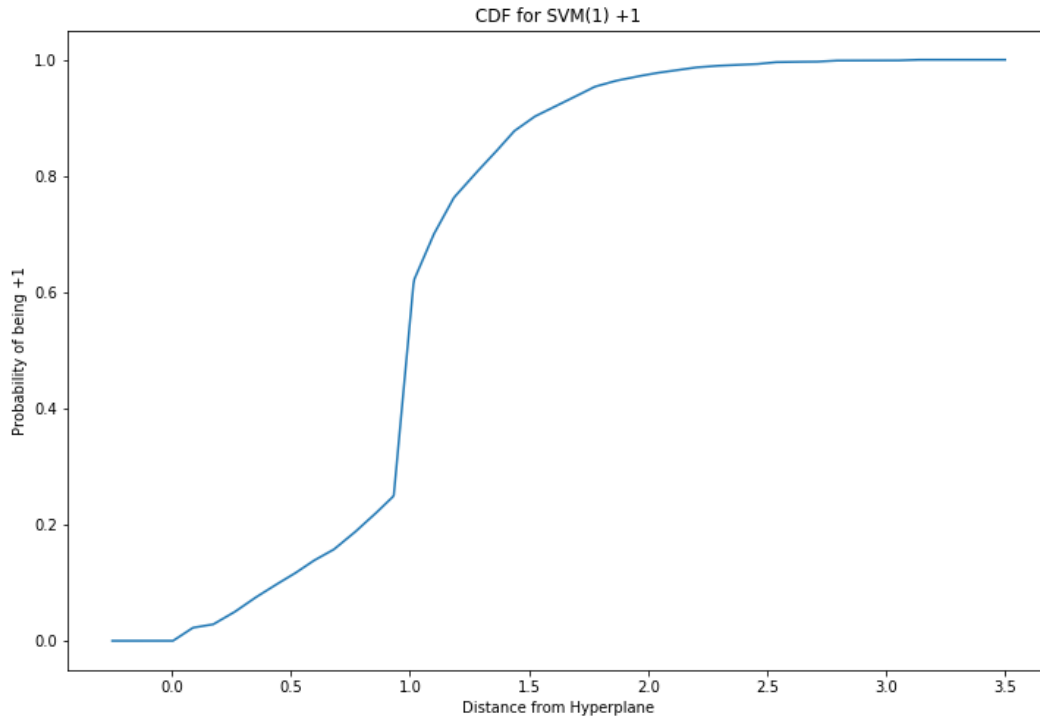


Fig 26: SVM<sub>1</sub> CDF of Correct +1 Prediction's for Train<sub>1</sub>

## SVM 2:

The distribution below is for +1 Class in SVM<sub>2</sub>, meaning SVM<sub>2</sub> says it is in Class 2 or high wattage.

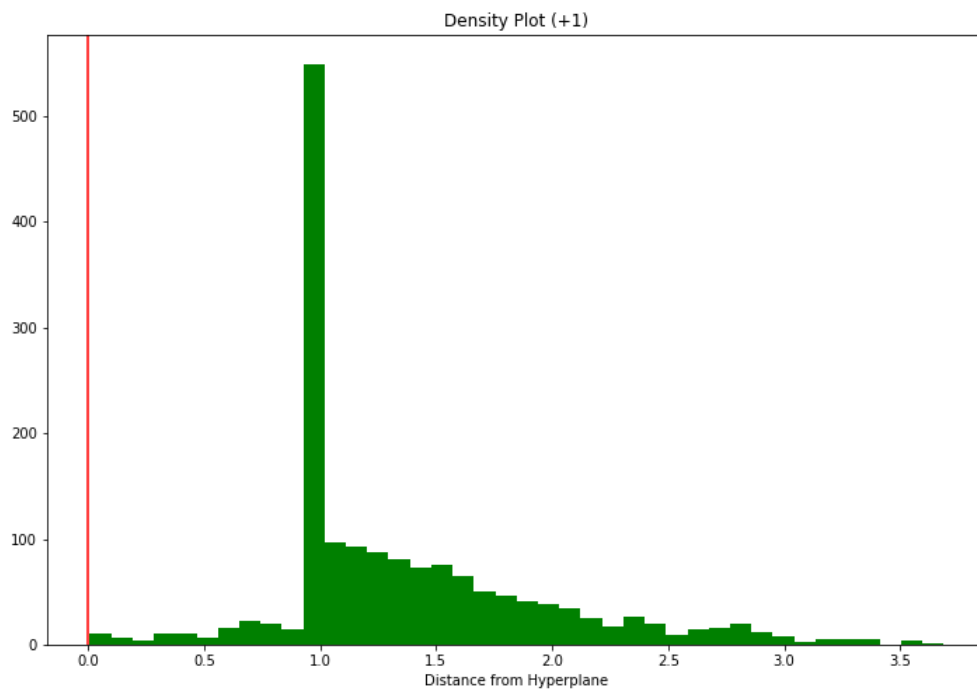


Fig 27: SVM<sub>2</sub> Histogram of Correct +1 Prediction's for Train<sub>2</sub>

The distribution graph above creates the  $CDF_2$  below which gives us our confidence for being in Class 3 based off of  $SVM_3$ .

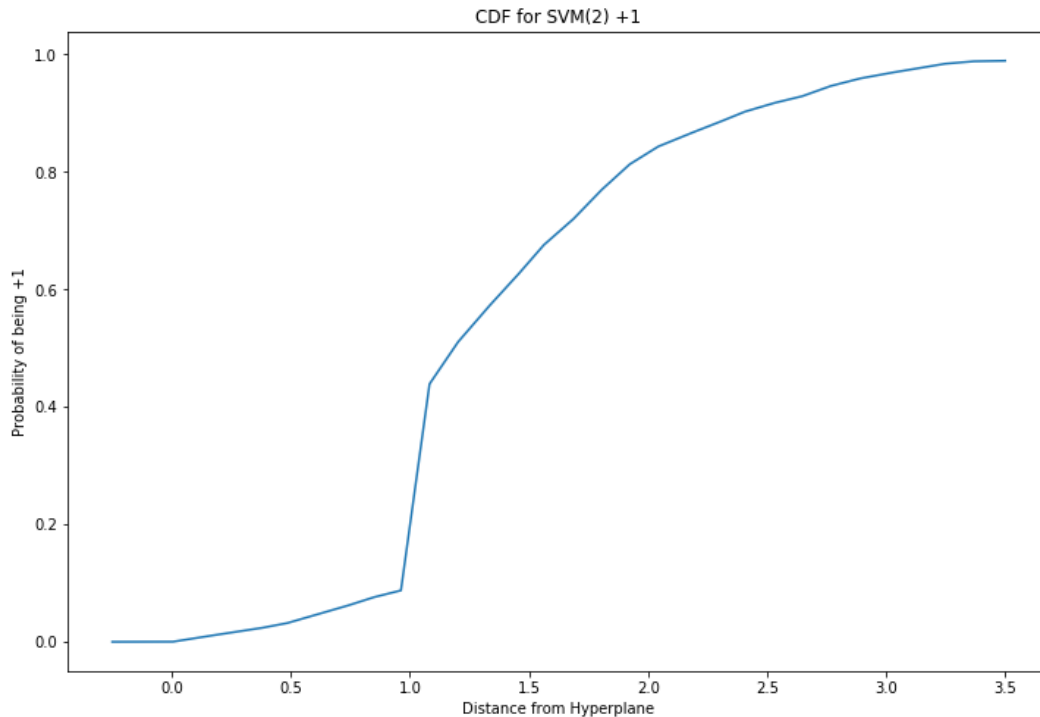


Fig 28:  $SVM_2$  CDF of Correct +1 Prediction's for Train<sub>2</sub>

$CDF_2$  gives us the following probabilities for the first 10 distance  $SVM_2$  gives us from the hyperplane.

	0	1	2	3	4	5	6	7	8	9
<b>SVM2</b>	0.085	0.198	0.197	0.432	0.478	0.777	0.769	0.650	0.477	0.197

Table 17:  $SVM_2$  Confidence in 1<sup>st</sup> Nine Observations of Train Set

As we can see, it is pretty confident about observation 3, 4, 5, 6, 7, and 8, but not very confident about the others.

### Total:

Combining the probabilities for these first 10 observations in the training set gives us the following table. You can see that in the first 10 cases it classified 2 of them wrong, giving us a 20% error rate in the first 10.

	SVM0	SVM1	SVM2	Prediction	Correct
0	0.160	0.547	0.085	1	1
1	0.448	0.134	0.198	0	2
2	0.448	0.056	0.197	0	2
3	0.000	0.119	0.432	2	2
4	0.000	0.059	0.478	2	2
5	0.000	0.000	0.777	2	2
6	0.000	0.000	0.769	2	2
7	0.000	0.000	0.650	2	2
8	0.000	0.000	0.477	2	2
9	0.013	0.054	0.197	2	2

Table 18: SVM's Confidence in 1<sup>st</sup> Nine Observations of Train Set and Terminal Predictions

Doing what we did for the first 10 observations of the training set for the entire training set gives us a percent confusion matrix like the one below.

TRAIN					
		PREDICTED			
ACTUAL	N(0) = 1855	Low Wattage (%)	Med Wattage (%)	High Wattage (%)	
	N(1) = 1684				
	N(2) = 1723				Sum (%)
	Low Wattage	82.16	16.50	1.35	100
	Medium Wattage	16.92	74.76	8.31	100
	High Wattage	2.73	11.90	85.37	100
		Lower Limit (%)	Percent Correct (%)	Upper Limit (%)	Margin
Low Wattage		81.12	82.16	83.19	1.03
Med. Wattage		73.59	74.76	75.94	1.17
High Wattage		84.42	85.37	86.33	0.95
Total Score		79.70	80.76	81.83	1.06

Table 19: Train Terminal Confusion Matrix and Limits

You can see off of this matrix, that by combining the three SVM's and using their CDF's, we get an overall score of  $80.76 \pm 1.06\%$ . This is a little worse than the individual confusion matrices which were giving us overall scores between 85% and 89%. The table below shows us the percent confusion matrix for the test set.

		TEST				
		PREDICTED				
ACTUAL	N(0) = 464	Low Wattage (%)	Med Wattage (%)	High Wattage (%)		
	N(1) = 422					
	N(2) = 431				Sum (%)	
	Low Wattage	71.34	25.86	2.80	100	
	Medium Watta	21.56	58.53	19.91	100	
	High Wattage	4.41	20.19	75.41	100	
		Lower Limit (%)	Percent Correct (%)	Upper Limit (%)	Margin	
	Low Wattage	68.89	71.34	73.78	2.44	
	Med. Wattage	55.87	58.53	61.19	2.66	
	High Wattage	73.08	75.41	77.73	2.33	
	Total Score	65.91	68.42	70.93	2.51	

Table 20: Test Terminal Confusion Matrix and Limits

Looking at this, we get an overall score that is  $68.42 \pm 2.51\%$ . The range of individual SVM scores ranged from 68% to 85%. I felt like this should of done a little better, but after looking at the confusion matrix we can see that our hang up resides in the classification of Class 2, with a score of  $58.53 \pm 2.66\%$ , and this makes sense because this is the class we were confident in.

One thing I noticed was that we lost about 10% in accuracy between the individual confusion matrices and the terminal confusion matrix. I feel like there may be a way to reduce this loss. There are other methods in doing this and running those methods and comparing them could lead to better results.

### Question 5: SVM classification using a polynomial kernel

The polynomial equation is as follow:

$$k(x_k, x) = (Coef_0 + \langle x_k^*, x \rangle)^d$$

*Equation 7: Polynomial Kernel Function*

I will follow the same process for polynomial as I did for radial, that being:

- Grid Search
- Manual Tune
- Individual Percent Confusion Matrices
- CDF's
- Terminal Confusion Matrix

### SVM<sub>0</sub>:

Looking at the results from the grid search below, it looks like degree 3 and degree 4 show similar results and the best test train scores looks to be with a Cost somewhere around 33.

Test vs. Train by degree's

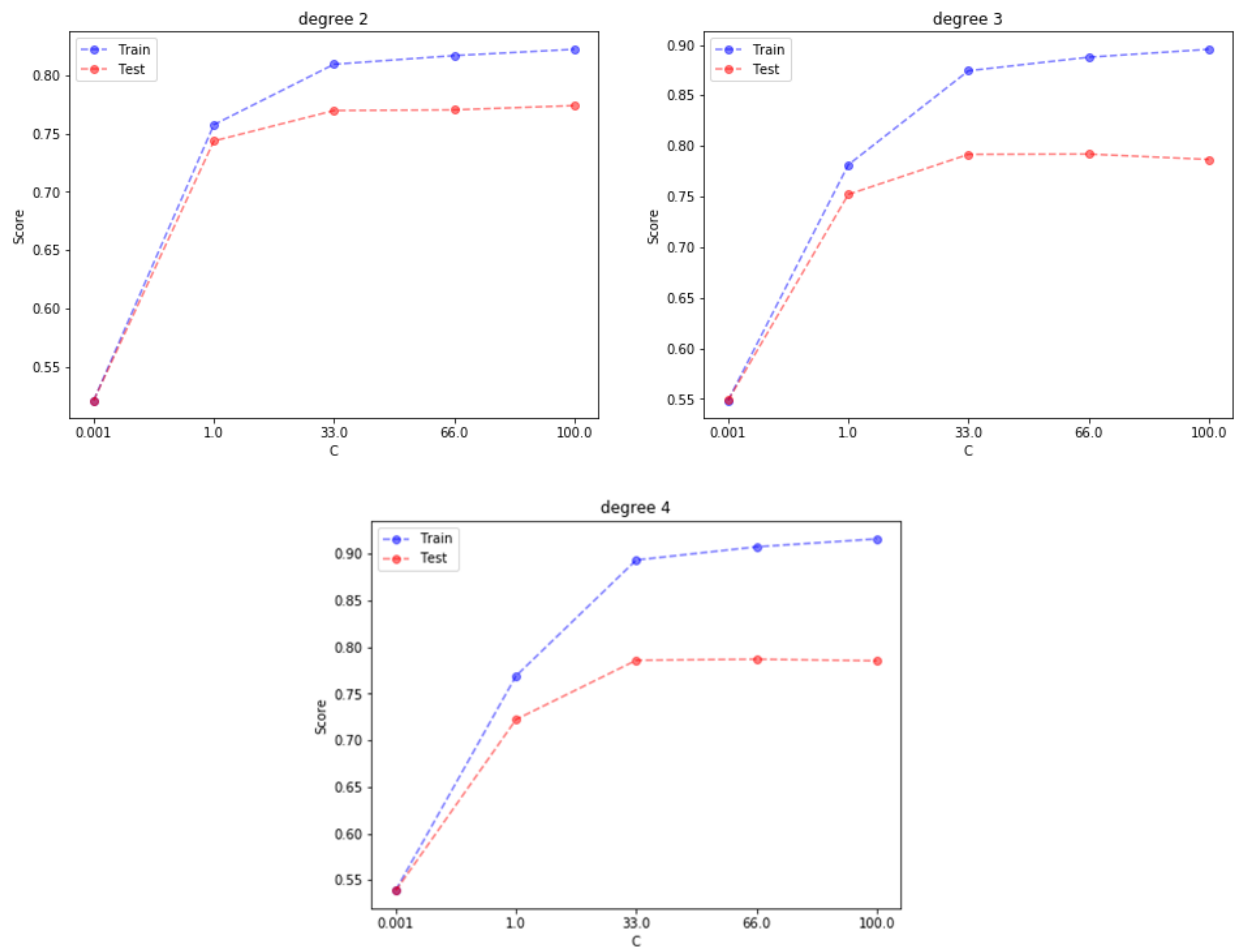


Fig 29: Grid Search Results for Parameters

Fixing the degree at 3 and look at a range of Costs around 30 gives us the following graph. It is easy to see that the major difference between the costs is the ratio of support vectors, and even that is not much, displayed by the small range of the y-axis.

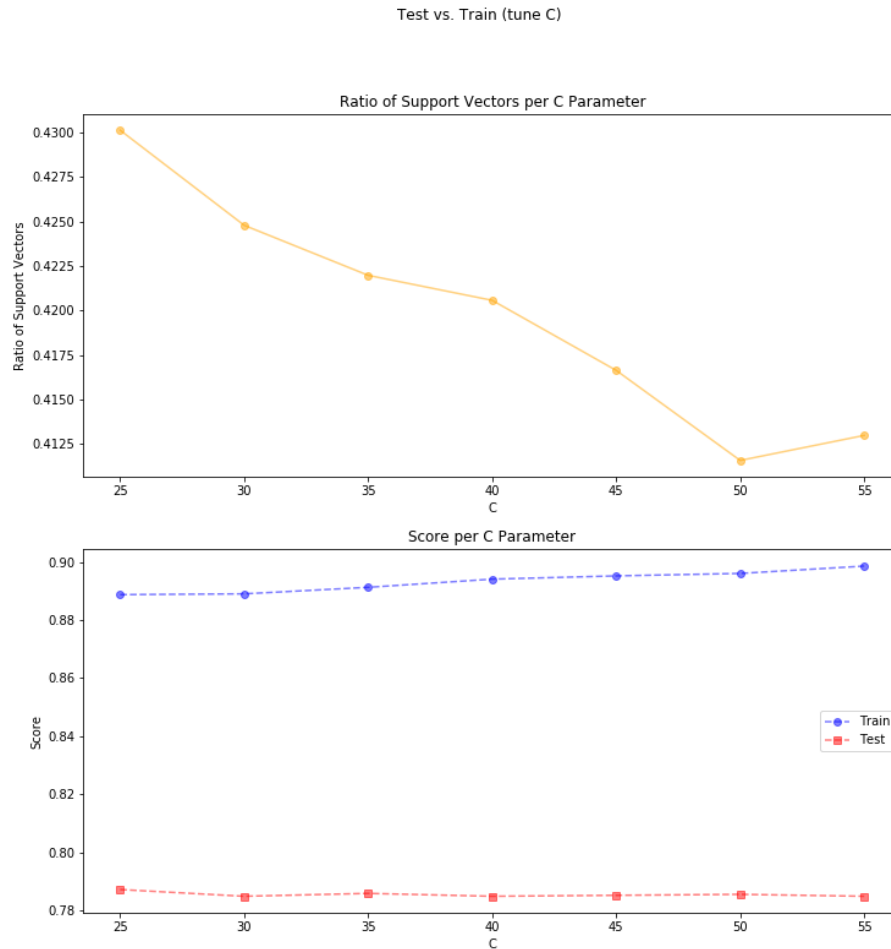


Fig 30: Ratio Support Vector and Train/Test Score relation for Manual Tune

From the tuning procedure I concluded with degree = 3 and Cost = 35.

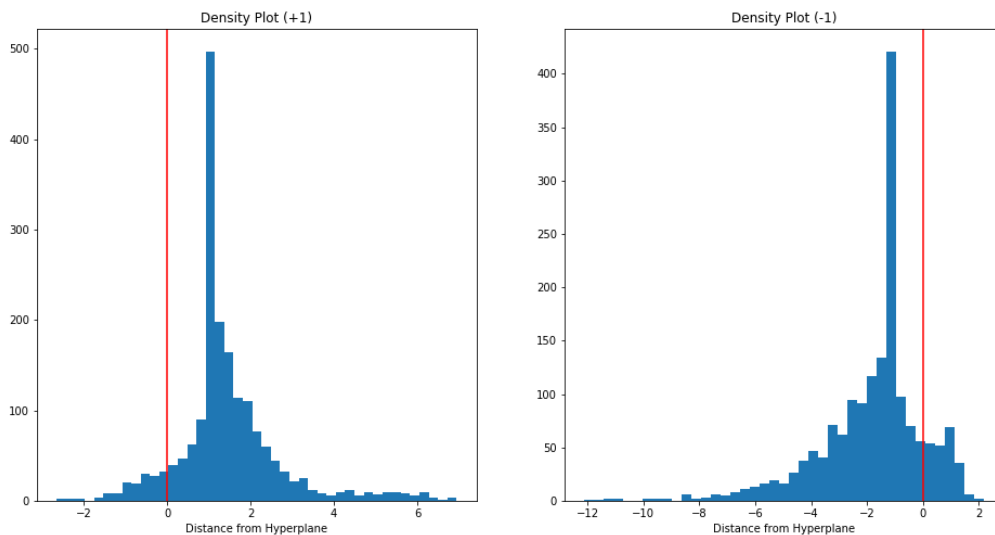
Using these parameters to train  $SVM_0$  with  $Train_0$ , I got the following training confusion matrix. As you can see from the matrix,  $SVM_0$  has an overall accuracy of about  $89 \pm 1.03\%$  which is not much different from radials results. The ratio of support vectors is slightly better than radials, but not significantly different.

TRAIN					
		Prediction			
N(+1) = 1855		All Others (%)	Low Wattage (%)		
N(-1) = 1703				Sum (%)	
Actual	All Others	79.59	20.41	100.00	
	Low Wattage	17.93	82.07	100.00	
		Lower Limit (%)	Percent Correct (%)	Upper Limit (%)	Margin (%)
All Others		84.89	86.02	87.16	1.14
Low Wattage		91.07	91.97	92.86	0.89
Total		87.97	89.00	90.02	1.03
Ratio SV		0.422			

Table 21: Train Confusion Matrix and Limits for SVM<sub>0</sub>

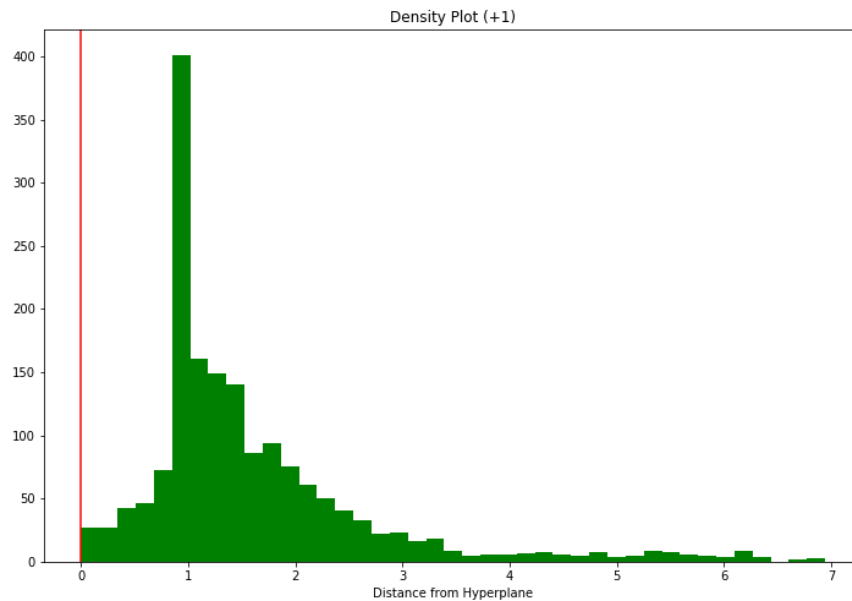
Looking at the density plots from the two classes in SVM<sub>0</sub> you can see that the x-axis has a little bit more of a range to it than the radial plots did, but the plots themselves are very similar. The range may be larger making the correct distances a little larger, but it also makes the error distances a little larger. In the end I feel the difference cancels out and it is practically the same plot as radials.

Frequency plots of Distance from Hyperplane

Fig 31: SVM<sub>0</sub> Histogram of Prediction's for Train<sub>0</sub>

Looking at the correctly classified +1's from SVM<sub>0</sub> gives the histogram below. From this histogram we make the CDF<sub>1</sub> that helps form our terminal predictions and confusion matrix.



Fig 32: SVM<sub>0</sub> Histogram of Correct +1 Prediction's for Train<sub>0</sub>

Using SVM<sub>0</sub> on the Test<sub>0</sub> set gives the following confusion matrix. Notice that the overall accuracy is  $80.20 \pm 1.42\%$ , showing no significant difference than the radial SVM<sub>0</sub>.

TEST					
		Prediction			
N(+1) = 464		All Others (%)	Low Wattage (%)		
N(-1) = 2557				Sum (%)	
Actual	All Others	77.86	22.14	100.00	
	Low Wattage	17.46	82.54	100.00	
		Lower Limit (%)	Percent Correct (%)	Upper Limit (%)	Margin (%)
All Others		76.38	77.86	79.35	1.48
Low Wattage		81.19	82.54	83.90	1.35
Total		78.78	80.20	81.62	1.42

Table 22: Test Confusion Matrix and Limits for SVM<sub>0</sub>

The histogram of predictions below does appear to be slightly different for -1 predictions. If you recall on the radial model, the histogram for -1 predictions a small peak on the incorrect side of the hyperplane and large peak on the correct side. This -1 histogram only shows a single peak, but the area of the density is the same on either side of the separating hyperplane, as shown by the similar Test<sub>0</sub> confusion matrices.

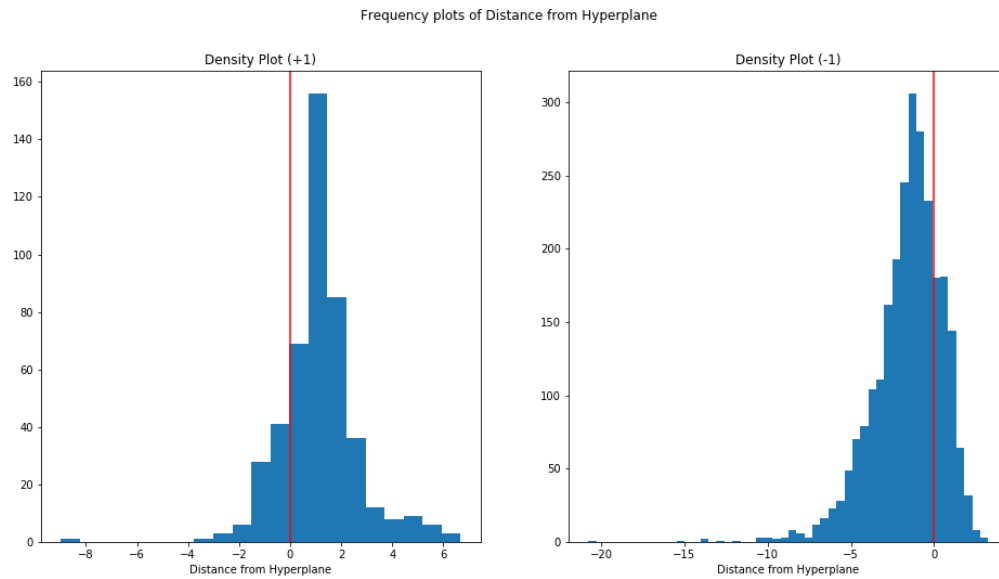
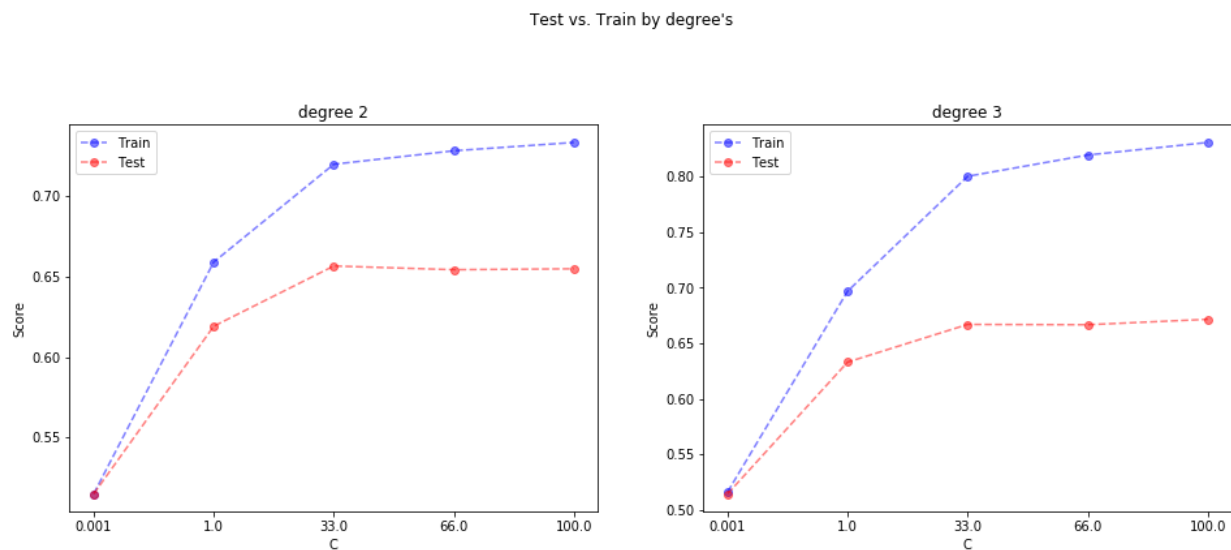


Fig 33: SVM<sub>0</sub> Histogram of Prediction's tor Test<sub>0</sub>

### SVM<sub>1</sub>:

The grid search for SVM<sub>1</sub> using Train<sub>1</sub> set was as follows. Looking at the graphs it appears the only difference lies in the spread between test and train's scores. The score itself wants us too focus in around 33.



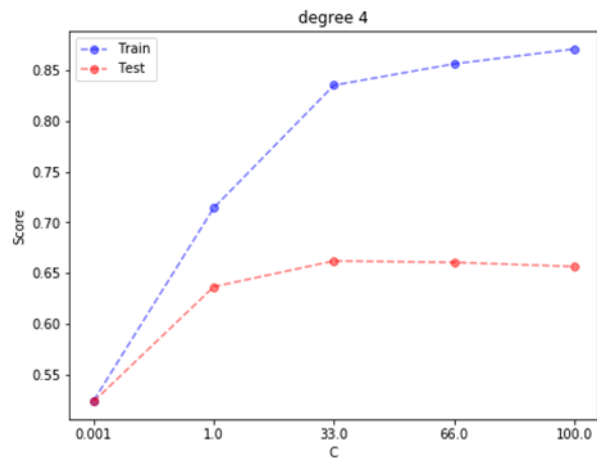


Fig 34: Grid Search Results for Parameters

I manually tuned with both degree 2 and degree 3 to see which one I wanted to go with. I liked the smaller spread between scores for degree two but the ratio of support vectors was almost 10% more so I went with fixing degree 3 and focusing on a range of Costs around 30 and got the results shown below.

Test vs. Train (tune C)

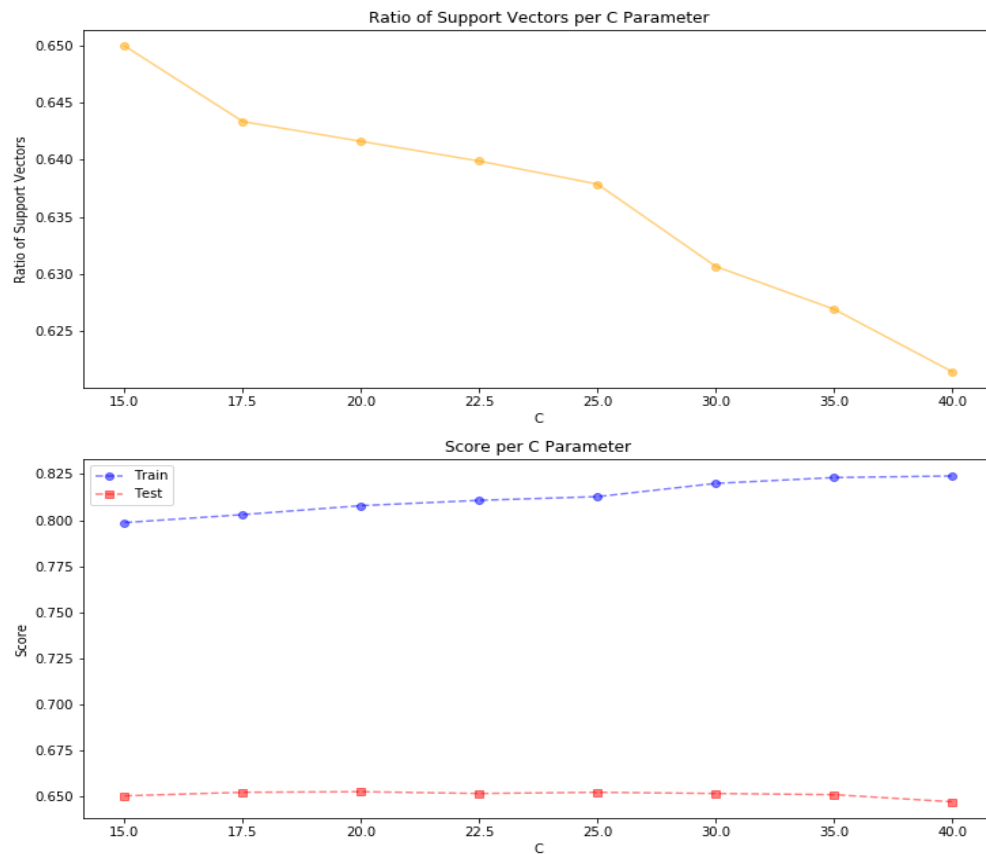


Fig 35: Ratio Support Vector and Train/Test Score relation for Manual Tune

From the tuning procedure I concluded with degree = 3 and Cost = 20.

With these parameters for fitting SVM<sub>1</sub> I got the following confusion matrix from the Train<sub>1</sub> set. There is an overall significant difference in this overall score of  $80.83 \pm 1.31\%$  at the 95% level for this polynomial model and an overall score of  $84.09 \pm 1.22\%$  for the radial model. But if you look at the test matrices there is no significant difference, and this is the one that matters in the end.

TRAIN					
		Prediction			
	N(+1) = 1684	All Others (%)	Med Wattage (%)		
	N(-1) = 1788			Sum (%)	
Actual	All Others	79.59	20.41	100.00	
	Med Wattage	17.93	82.07	100.00	
		Lower Limit (%)	Percent Correct (%)	Upper Limit (%)	Margin (%)
	All Others	78.25	79.59	80.93	1.34
	Med Wattage	80.79	82.07	83.34	1.28
	Total	79.52	80.83	82.14	1.31
	Ratio SV	0.641			

Table 23: Train Confusion Matrix and Limits for SVM<sub>1</sub>

SVM<sub>1</sub>'s predictions of Train<sub>1</sub> made the following histograms. Again, not much difference here.

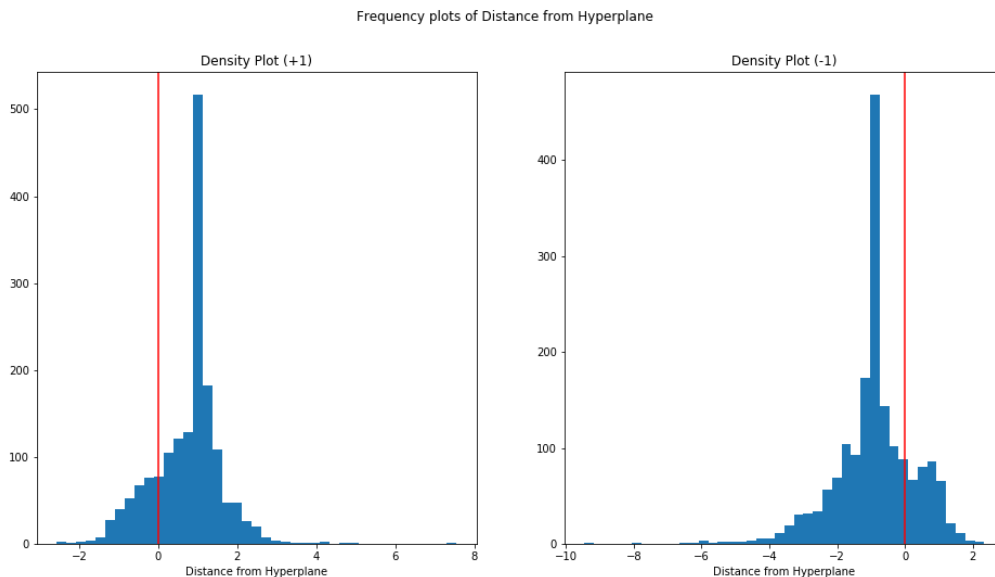


Fig 36: SVM<sub>1</sub> Histogram of Prediction's for Train<sub>1</sub>

The histogram of SVM<sub>1</sub>'s correct +1 predictions may look like the peak is closer to the separating hyperplane, but this is just because the range is greater and the scale for x-axis is larger. Using this histogram, we created the CDF<sub>1</sub> that will help in our terminal predictions and confusion matrix.

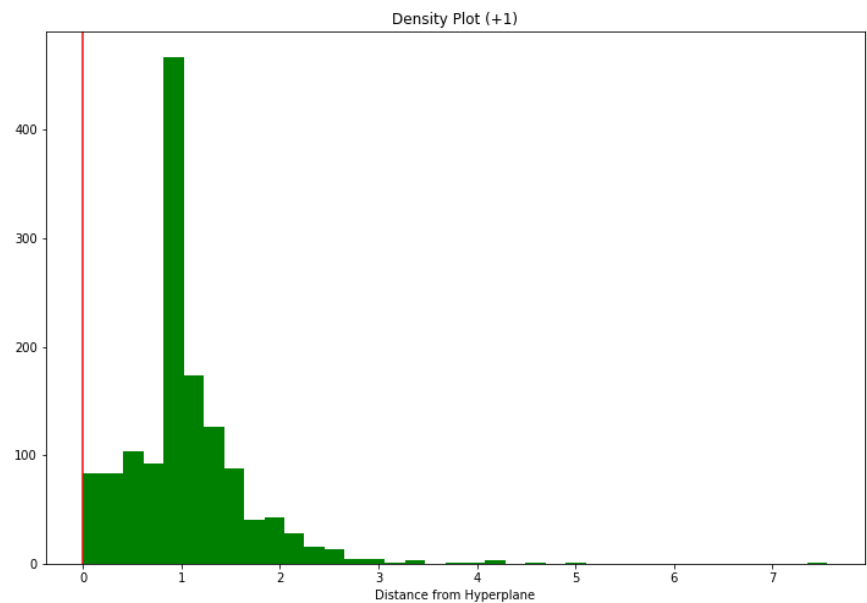


Fig 37: SVM<sub>1</sub> Histogram of Correct +1 Prediction's for Train<sub>1</sub>

Running SVM<sub>1</sub> on Test<sub>1</sub> gave the following test confusion matrix. There is no significant difference between the radial and polynomial models of SVM<sub>1</sub>, their estimated overall scores are only 0.5% different.

TEST					
		Prediction			
Actual	N(+1) = 422	All Others (%)	Med Wattage (%)	Sum (%)	
	N(-1) = 2685				
	Med Wattage	64.43	35.57	100.00	
	All Others	29.62	70.38	100.00	
		Lower Limit (%)	Percent Correct (%)	Upper Limit (%)	Margin (%)
All Others		62.75	64.43	66.12	1.68
Med Wattage		68.77	70.38	71.98	1.61
Total		65.76	67.41	69.05	1.65

Table 24: Test Confusion Matrix and Limits for SVM<sub>1</sub>

The SVM<sub>1</sub> predictions of Test<sub>1</sub> shown below give me similar feelings and don't inspire much confidence in this particular model.

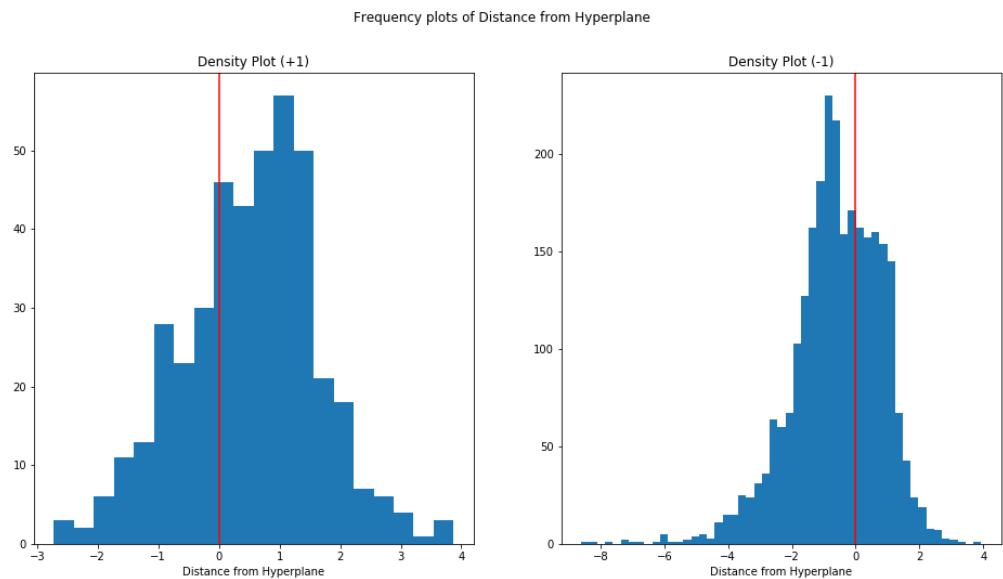
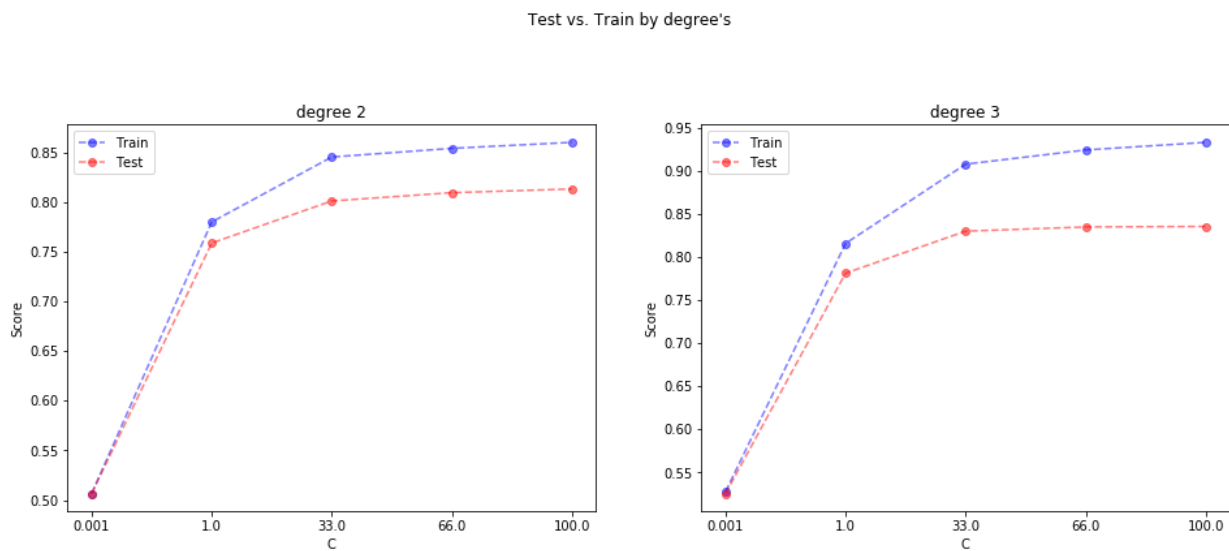


Fig 38: SVM<sub>1</sub> Histogram of Prediction's tor Test<sub>1</sub>

**SVM<sub>2</sub>:**

Looking at the grid search tables below for SVM<sub>2</sub>'s parameters, we get similar areas of interest with a degree of 3 and cost somewhere in vicinity of 30.



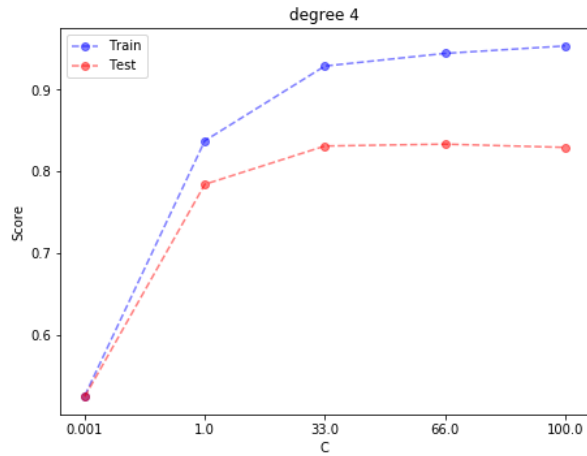


Fig 39: Grid Search Results for Parameters

Fixing the degree parameter to 3 and focusing on a cost around 30, I manually tuned for Cost and got the following results. There was not much difference between the different costs, except in the ratio of support vectors, but looking at the y-axis you can see that this is not even that much.

Test vs. Train (tune C)

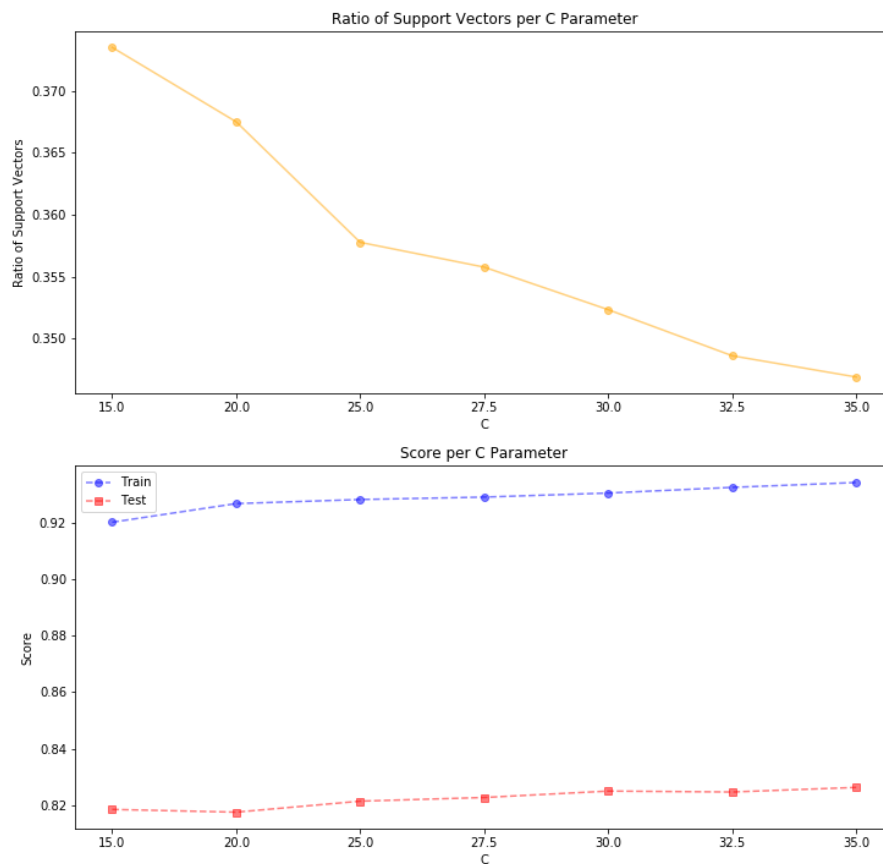


Fig 40: Ratio Support Vector and Train/Test Score relation for Manual Tune

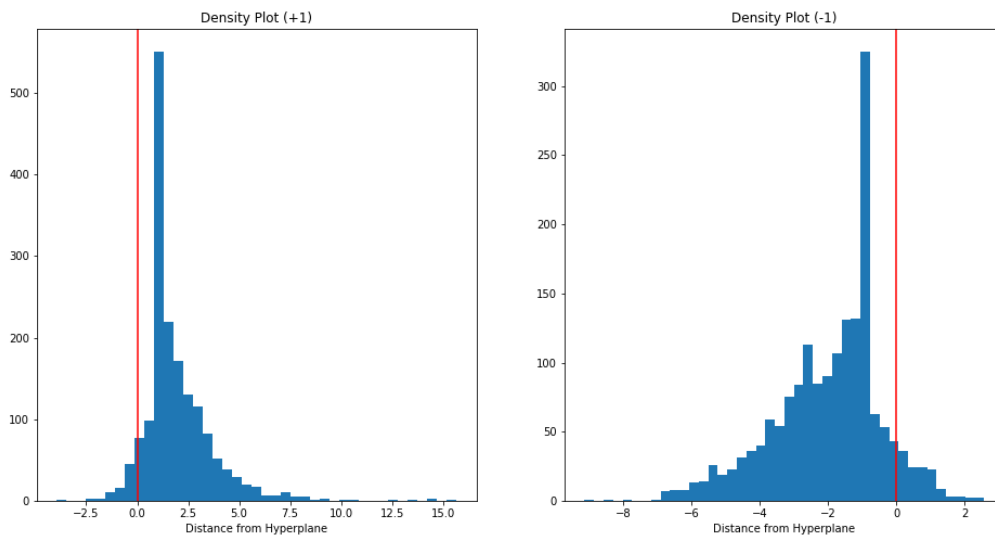
From the tuning procedure I concluded with degree = 3 and Cost = 35 just because it had the best test score and lowest ratio of support vectors. Using those parameters to train SVM<sub>2</sub> on Train<sub>1</sub> gave me the following confusion matrix for Train<sub>1</sub>. Looking at the results of this confusion matrix there is not really in significant difference between radials model. This model seemed to do a little worse all around, but not significantly. There is about a 4% reduction in support vectors with using this polynomial model compared to the radial model.

TRAIN					
		Prediction			
Actual	N(+1) = 1723	All Others (%)	High Wattage (%)	Sum (%)	
	N(-1) = 1769				
	All Others	92.43	7.57	100.00	
	High Wattage	5.57	94.43	100.00	
		Lower Limit (%)	Percent Correct (%)	Upper Limit (%)	Margin (%)
All Others		91.55	92.43	93.30	0.88
High Wattage		93.67	94.43	95.19	0.76
Total		92.60	93.43	94.25	0.82
Ratio SV		0.347			

Table 25: Train Confusion Matrix and Limits for SVM<sub>2</sub>

In the density graphs from the SVM<sub>1</sub> model on Train<sub>1</sub> below, it appears that the slope for quantity of distances from separating hyperplane's starts to increase sooner than in the previous model. This makes me feel less confident than I felt in the other model.

Frequency plots of Distance from Hyperplane

Fig 41: SVM<sub>2</sub> Histogram of Prediction's for Train<sub>2</sub>



The histogram of SVM<sub>2</sub>'s correct +1 predictions on Train<sub>2</sub> is shown below. This is used to make the CDF<sub>2</sub> that helps make our terminal predictions and confusion matrix.

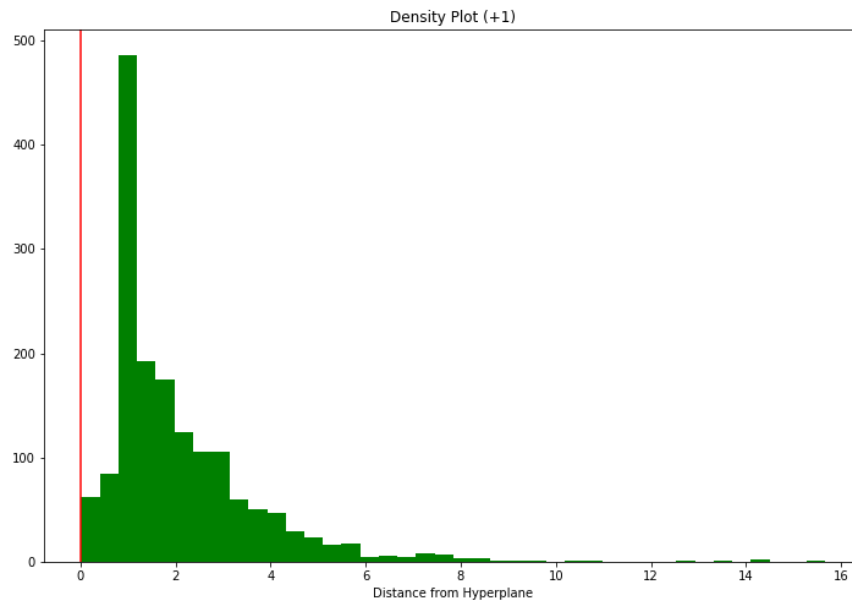


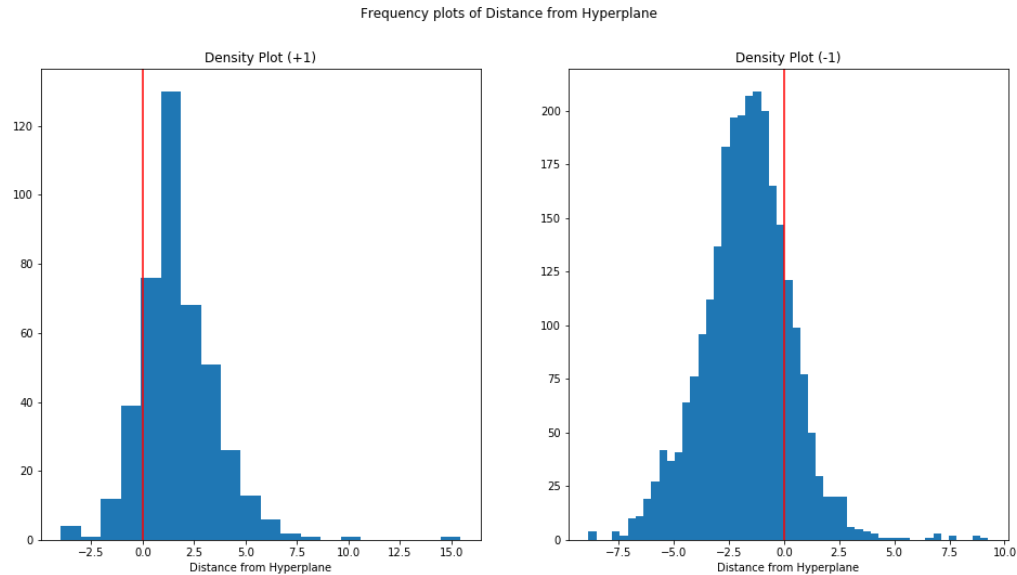
Fig 42: SVM<sub>2</sub> Histogram of Correct +1 Prediction's for Train<sub>2</sub>

Running SVM<sub>2</sub> on Test<sub>2</sub> gave the following test confusion matrix. There is no significant difference between the radial and polynomial models of SVM<sub>2</sub>.

TEST					
Actual		Prediction			
	N(+1) = 431	All Others (%)	High Wattage (%)	Sum (%)	
	N(-1) = 2656				
	All Others	82.12	17.88	100.00	
	High Wattage	14.15	85.85	100.00	
		Lower Limit (%)	Percent Correct (%)	Upper Limit (%)	Margin (%)
	All Others	80.76	82.12	83.47	1.35
	High Wattage	84.62	85.85	87.08	1.23
	Total	82.69	83.98	85.28	1.29

Table 26: Test Confusion Matrix and Limits for SVM<sub>2</sub>

The following histogram displays the distances from the separating hyperplane for predictions by SVM<sub>2</sub> on Test<sub>2</sub>.

Fig 43: SVM<sub>2</sub> Histogram of Prediction's for Test<sub>2</sub>

### Terminal Confusion Matrix

Using all of the CDF's from all the polynomial SVM models, we were able to make terminal predictions of 0, 1, 2 for Low Wattage, Med Wattage, High Wattage. Doing this for the full training set resulted in the following terminal matrix. This Confusion Matrix gives us an overall accuracy of  $78.10 \pm 1.12\%$  for the polynomial SVM's and the radial SVM's gave us an overall accuracy of  $80.76 \pm 1.06$ , meaning the radial SVM's gave us significantly better result, but not by much and the real data set we are interested in is the results from the test set. It looks like both Low Wattage and Med. Wattage are the factors that were significantly less accurate reducing the overall score.

TRAIN					
		PREDICTED			
ACTUAL	N(0) = 1855	Low Wattage (%)	Med Wattage (%)	High Wattage (%)	Sum (%)
	N(1) = 1684				
	N(2) = 1723				
	Low Wattage	79.46	18.76	1.78	100
	Medium Wattage	18.23	71.08	10.69	100
	High Wattage	3.13	13.12	83.75	100
		Lower Limit (%)	Percent Correct (%)	Upper Limit (%)	Margin
Low Wattage		78.37	79.46	80.55	1.09
Med. Wattage		69.86	71.08	72.31	1.23
High Wattage		82.75	83.75	84.75	1.00
Total Score		76.98	78.10	79.21	1.12

Table 27: Train Terminal Confusion Matrix and Limits for Terminal Predictions

Looking at the terminal confusion matrix for the full Test set below, we see that there is no significant difference between the polynomial model,  $68.37 \pm 2.51$ , and the radial model,  $68.42 \pm 2.51$ . You know they are close when the margins are the same.

		TEST				
		PREDICTED				
ACTUAL	N(0) = 464	Low Wattage (%)	Med Wattage (%)	High Wattage (%)		
	N(1) = 422					
	N(2) = 431				Sum (%)	
	Low Wattage	70.47	25.86	3.66	100	
	Medium Watta	21.80	58.77	19.43	100	
	High Wattage	5.10	19.03	75.87	100	
		Lower Limit (%)	Percent Correct (%)	Upper Limit (%)	Margin	
		Low Wattage	68.01	70.47	72.94	2.46
		Med. Wattage	56.11	58.77	61.43	2.66
		High Wattage	73.56	75.87	78.18	2.31
		Total Score	65.86	68.37	70.88	2.51

Table 28: Test Terminal Confusion Matrix and Limits for Terminal Predictions

## Conclusion

Since there was no significant difference between the test set of the two different models, I looked at the SVM's prediction histograms to help guide my decision. The densities for the radial SVM's made me feel a little more comfortable because I felt that the density of observations close to the hyperplane was lower until it got to its expected distance.

One thing I would have liked to do, would be to look at the incorrectly classified observations and look at the top most confident incorrectly classified observations. I would like to compare the explanatory variables of these to the explanatory variables of the top confident correctly classified observations for each class. This may give me insight as to what variable is giving me problems, particularly in class two.

I feel like the model does okay for classification, but I believe there is a random factor in the data that this model is not good at handling, and that is the Wattage used when people are home. If this were a neural network it would learn this relationship and account for it. But this model does not learn that way. If I were to do this again, instead of reducing the number of observations by the expanding the time steps, I would eliminate all of the observations with lights on. This would give me a dataset with only observations while the people in the home were away from home or asleep. I believe this would reduce the variability of the Wattage in the data.

## Times

Below is a table of operation times for running the grid search, manual tuning, fit and train results, and test set results. Looking at the totals of the columns you can see that the polynomial was faster on the grid search but the radial model was faster on the manual tuning, fit/training results, and test set

results. The RBF in all looks longer to come with my finals SVM's for each class, but this is because there were two parameters to manually tune.

RBF Model Timings						
	Grid 1	Tune 1	Tune 2	Train	Test	Total (min)
<b>SVM0</b>	232.63	9.93	11.11	2.16	1.37	4.29
<b>SVM1</b>	230.86	12.35	13.61	2.59	1.64	4.35
<b>SVM2</b>	213.88	8.41	9.19	1.73	1.58	3.91
<b>Total (min)</b>	11.29	0.51	0.57	0.11	0.08	12.55

Table 29: Times for Radial Tuning and Results

Polynomial					
	Grid 1	Tune 1	Train	Test	Total (min)
<b>SVM0</b>	179.92	18.98	3.53	0.95	3.39
<b>SVM1</b>	242.15	25.16	3.80	1.36	4.54
<b>SVM2</b>	193.44	14.43	2.81	0.97	3.53
<b>Time (min)</b>	10.26	0.98	0.17	0.05	11.46

Table 30: Times for Polynomial Tuning and Results

Dustin Vasquez - 1917828

```
# -*- coding: utf-8 -*-  
"""
```

Created on Wed Nov 20 18:48:19 2019

```
@author: Dustin  
"""
```

```
import numpy as np  
import pandas as pd  
import matplotlib.pyplot as plt  
import time as time
```

```
#####
```

```
#Import Data/Describe Data
```

```
#####
```

```
fname = \  
r'C:\Users\Dustin\Desktop\Masters Program\Fall Semester\aaaaStatistical Learning and Data  
mining\Homework and Readings\Homework\HW4\data2.csv'  
fname1 = \  
r'C:\Users\Dustin\Desktop\Masters Program\Fall Semester\aaaaStatistical Learning and Data  
mining\Homework and Readings\Homework\HW4\data2_appliance.csv'  
data = pd.read_csv(fname)
```

```
#Class and Meanings
```

```
class_name = {0:'Low Wattage',  
              1:'Medium Wattage',  
              2:'High Wattage'}
```

```
#plot discrete frequency
```

```
fig, ax = plt.subplots(figsize = (12,8))  
ax.hist(data['month'], align = 'left',rwidth = .95, bins = [1,2,3,4,5,6])  
ax.set_title('Frequency of Months')  
ax.set_xlabel('Month')
```

```
#####
```

```
#Organize the Data for running SVM
```

```
#####
```

```
#Test and Train
```

```
import test_train as tt
```

```
#set it up
```

```
cont = tt.test_train()
```

```
#Organize Data (1st step)
cont.set_xy_class(list(data.columns), y_columns = ['Class', 'Wattage'], column = 'Class')

#get dummies
#data_final = cont.get_dummies(data, dummies = ['month'])

#print description before
data.describe()
#      Class  Wattage ...  Visibility  Tdewpoint
#count 6579.000000 6579.000000 ... 6579.000000 6579.000000
#mean   0.974920 101.531134 ... 38.329685   3.761663
#std    0.824236 93.047128 ... 11.664370   4.194497
#min    0.000000 20.000000 ... 1.000000  -6.550000
#25%    0.000000 50.000000 ... 29.166667   0.900000
#50%    1.000000 63.333333 ... 40.000000   3.433333
#75%    2.000000 110.000000 ... 40.000000   6.566667
#max    2.000000 796.666667 ... 66.000000  15.400000

#Get standardized
data_final, stdz_params = cont.get_standardized(data, classification = True)

#print description after
data_final.describe()
#      Class  Wattage ...  Visibility  Tdewpoint
#count 6579.000000 6579.000000 ... 6.579000e+03 6.579000e+03
#mean   0.974920 101.531134 ... 6.361634e-16 -1.927829e-16
#std    0.824236 93.047128 ... 1.000076e+00 1.000076e+00
#min    0.000000 20.000000 ... -3.200561e+00 -2.458566e+00
#25%    0.000000 50.000000 ... -7.856159e-01 -6.822942e-01
#50%    1.000000 63.333333 ... 1.432089e-01 -7.828234e-02
#75%    2.000000 110.000000 ... 1.432089e-01 6.687850e-01
#max    2.000000 796.666667 ... 2.372389e+00 2.774879e+00

#Greater than two classes
data_final = cont.greater_two(data_final)

#get test and train
train, test = cont.get_test_train(data = data_final, set_seed = 229)

#####
#SVM
#####
from SVM import SVM
```

```
mySVM = SVM()
```

```
#largest classes
```

```
data_final.groupby('Class').sum().iloc[:,0:3]
```

```
#    0_vs_all  2_vs_all  1_vs_all
```

```
#Class
```

```
#0      2319   -2319   -2319
```

```
#1     -2106   -2106    2106
```

```
#2     -2154    2154   -2154
```

```
class_to_num = {'SVM0' : 0,
```

```
                'SVM1' : 1,
```

```
                'SVM2' : 2
```

```
}
```

```
train.groupby('Class', axis = 0).count()['0_vs_all']
```

```
#Class
```

```
#0   1855
```

```
#1   1684
```

```
#2   1723
```

```
test.groupby('Class', axis = 0).count()['0_vs_all']
```

```
#Class
```

```
#0   464
```

```
#1   422
```

```
#2   431
```

```
##### Get Proportions for each class #####
```

```
indx_0 = np.random.choice(a = np.array(train[train['Class']==0].index),
```

```
                           size = int(.5*len(train[train['Class']==0].index)), replace = False)
```

```
indx_2 = np.random.choice(a = np.array(train[train['Class']==2].index),
```

```
                           size = int(.5*len(train[train['Class']==2].index)), replace = False)
```

```
indx_1 = np.random.choice(a = np.array(train[train['Class']==1].index),
```

```
                           size = int(.5*len(train[train['Class']==1].index)), replace = False)
```

```
#get the training set with 0
```

```
index_for_0 = list(indx_1)+list(indx_2) + list(train[train['Class']==0].index)
```

```
print('index:', len(index_for_0), '\n uniques:', len(np.unique(np.array(index_for_0))))
```

```
train_0 = train.iloc[index_for_0].sample(frac = 1, axis = 'index').reset_index(drop = True)
```

```
test_0 = pd.concat([test, train.drop(index_for_0, axis = 0)], axis = 0).sample(frac = 1, axis =  
'index').reset_index(drop = True)
```

```
print('\nTrain 0: \n{}'.format(train_0.groupby(by = 'Class', axis = 0).count()['0_vs_all']))
```

```
print('\nTest 0: \n{}'.format(test_0.groupby(by = 'Class', axis = 0).count()['0_vs_all']))
```

#get the training set with 1

```
index_for_1 = list(indx_0)+list(indx_2) + list(train[train['Class']==1].index)
print('index:', len(index_for_1), '\n uniques:', len(np.unique(np.array(index_for_1))))
train_1 = train.iloc[index_for_1].sample(frac = 1, axis = 'index').reset_index(drop = True)
test_1 = pd.concat([test, train.drop(index_for_1, axis = 0)], axis = 0).sample(frac = 1, axis =
'index').reset_index(drop = True)
print('\nTrain 1: \n{}'.format(train_1.groupby(by = 'Class', axis = 0).count()['0_vs_all']))
print('\nTest 1: \n{}'.format(test_1.groupby(by = 'Class', axis = 0).count()['0_vs_all']))
```

#get the training set with 2

```
index_for_2 = list(indx_0)+list(indx_1) + list(train[train['Class']==2].index)
print('index:', len(index_for_2), '\n uniques:', len(np.unique(np.array(index_for_2))))
train_2 = train.iloc[index_for_2].sample(frac = 1, axis = 'index').reset_index(drop = True)
test_2 = pd.concat([test, train.drop(index_for_2, axis = 0)], axis = 0).sample(frac = 1, axis =
'index').reset_index(drop = True)
print('\nTrain 2: \n{}'.format(train_2.groupby(by = 'Class', axis = 0).count()['0_vs_all']))
print('\nTest 2: \n{}'.format(test_2.groupby(by = 'Class', axis = 0).count()['0_vs_all']))
```

##### Optimize Gamma and Cost for class 0 #####

```
mySVM.set_kernel(kernel = 'rbf')
time_SVM0 = {}
```

##Work on optimization

```
a = time.time()
param_grid = {'gamma': [.00001, .001, .1, 1],
              'C': [.00001, .1, 20, 60, 100]
              }
```

#grid search

```
cv, best_estimator, b_score, b_params = mySVM.gridsearch(train_0[cont._test_train__x_columns],
train_0['0_vs_all'], param_grid = param_grid, random_state = 229,
cv=10, return_train_score=True, it = 0.1)
```

```
time_SVM0['Grid Search'] = [time.time()-a]
```

# Manual Tune (gamma = .01)

```
a = time.time()
```

```
mySVM.set_kernel(kernel = 'rbf', gamma = .01)
tune0_scores, n_sv0 = mySVM.tune_svm(train_0[cont._test_train__x_columns],
train_0['0_vs_all'], test_0[cont._test_train__x_columns], test_0['0_vs_all'],
C=1, random_state = 229, tune = 'C',
params = [25, 30, 35, 40, 45, 50, 55], it = 0.1)
```



Dustin Vasquez - 1917828

```
time_SVM0['Manual Tune'] = [time.time()-a]
```

```
# Manual Tune (Cost = 50)
```

```
a = time.time()
```

```
mySVM.set_kernel(kernel = 'rbf', gamma = .01)
```

```
tune0_scores, n_sv0 = mySVM.tune_svm(train_0[cont._test_train__x_columns],  
                                     train_0['0_vs_all'], test_0[cont._test_train__x_columns], test_0['0_vs_all'],  
                                     C=50, random_state = 229, tune = 'gamma',  
                                     params = [.005, .0075, .01, .025, .05, .075, .1, .125], it = 0.2)
```

```
time_SVM0['Manual Tune'] += [time.time()-a]
```

```
##### Optimize Gamma and Cost for class 1 #####
```

```
mySVM.set_kernel(kernel = 'rbf')
```

```
time_SVM1 = {}
```

```
##Work on optimization
```

```
a = time.time()
```

```
param_grid = {'gamma': [.00001, .001, .1, 1],  
              'C': [.00001, .1, 20, 60, 100]  
              }
```

```
#grid search
```

```
cv, best_estimator, b_score, b_params = mySVM.gridsearch(train_1[cont._test_train__x_columns],  
                                                         train_1['1_vs_all'], param_grid = param_grid, random_state = 229,  
                                                         cv=10, return_train_score=True, it = 1.1)
```

```
time_SVM1['Grid Search'] = [time.time()-a]
```

```
# Manual Tune (gamma = .1)
```

```
a = time.time()
```

```
mySVM.set_kernel(kernel = 'rbf', gamma = .1)
```

```
tune1_scores, n_sv1 = mySVM.tune_svm(train_1[cont._test_train__x_columns],  
                                     train_1['1_vs_all'], test_1[cont._test_train__x_columns], test_1['1_vs_all'],  
                                     C=1, random_state = 229, tune = 'C',  
                                     params = [10, 15, 17.5, 20, 22.5, 25, 30, 35], it = 1.1)
```

```
time_SVM1['Manual Tune'] = [time.time()-a]
```

```
# Manual Tune (Cost = 22.5)
```

```
a = time.time()
```

```
mySVM.set_kernel(kernel = 'rbf', gamma = .1)
tune1_scores, n_sv1 = mySVM.tune_svm(train_1[cont._test_train__x_columns],
                                     train_1['1_vs_all'], test_1[cont._test_train__x_columns], test_1['1_vs_all'],
                                     C=22.5, random_state = 229, tune = 'gamma',
                                     params = [.01, .025, .05, .075, .1, .125, .150, .175], it = 1.2)

time_SVM1['Manual Tune'] += [time.time()-a]

##### Optimize Gamma and Cost for Class 2 #####
##Work on optimization
mySVM.set_kernel(kernel = 'rbf')
time_SVM2 = {}

#grid search
a = time.time()
param_grid = {'gamma': [.00001, .001, .1, 1],
              'C': [.00001, .1, 20, 60, 100]
              }

cv, best_estimator, b_score, b_params = mySVM.gridsearch(train_2[cont._test_train__x_columns],
                                                         train_2['2_vs_all'], param_grid = param_grid, random_state = 229,
                                                         cv=10, return_train_score=True, it = 2.1)

time_SVM2['Grid Search'] = [time.time()-a]

# Manual Tune (gamma = .1)
a = time.time()

mySVM.set_kernel(kernel = 'rbf', gamma = .1)
tune2_scores, n_sv2 = mySVM.tune_svm(train_2[cont._test_train__x_columns],
                                     train_2['2_vs_all'], test_2[cont._test_train__x_columns], test_2['2_vs_all'],
                                     C=1, random_state = 229, tune = 'C',
                                     params = [10, 15, 17.5, 20, 22.5, 25, 30, 35], it = 2.1)

time_SVM2['Manual Tune'] = [time.time()-a]

# Manual Tune (Cost = 20)
a = time.time()

mySVM.set_kernel(kernel = 'rbf', gamma = .1)
tune2_scores, n_sv2 = mySVM.tune_svm(train_2[cont._test_train__x_columns],
                                     train_2['2_vs_all'], test_2[cont._test_train__x_columns], test_2['2_vs_all'],
                                     C=20, random_state = 229, tune = 'gamma',
```

```
params = [.01, .025, .05, .075, .1, .125, .150, .175], it = 2.2)

time_SVM2['Manual Tune'] += [time.time()-a]

#####
#3 SVM's, 3 Confusion Matrixes/sets of histograms
#####

#####   SVM 0 vs All (0 is low wattage) #####
a = time.time()

#set Kernel
gamma = .025
C = 50
svm0 = SVM()
svm0.set_kernel(kernel = 'rbf', gamma = gamma)

#training set
r_sv0, sv0, y_predict0, score0, conf_int0, dist_hp0, model0 = \
    svm0.run_svm(train_0[cont._test_train__x_columns], train_0['0_vs_all'], C=C, random_state = 229)

train_conf0, train_per_conf0, train_limits0, train_confidence0 =
svm0.get_confusions(train_0['0_vs_all'], y_predict0)

#histogram
hist_dist0 = svm0.get_hist(dist_hp0, train_0['0_vs_all'], it = 0.1)

time_SVM0['Train'] = [time.time()-a]
a = time.time()

#test set
test_predict0, test_score0, test_conf_int0, test_dist_hp0 = \
    svm0.run_svm(test_0[cont._test_train__x_columns], test_0['0_vs_all'], C=C, random_state = 229,
        train = 'no', model = model0)

test_conf0, test_per_conf0, test_limits0, test_confidence0 = svm0.get_confusions(test_0['0_vs_all'],
test_predict0, conf_desired = .95)

#histogram
test_hist_dist0 = svm0.get_hist(test_dist_hp0, test_0['0_vs_all'], it = 0.2)

time_SVM0['Test'] = [time.time()-a]
#####   SVM 1 vs All (1 is med wattage) #####
```

Dustin Vasquez - 1917828

```
a = time.time()

#set Kernel
gamma = .05
C = 22.5
svm1 = SVM()
svm1.set_kernel(kernel = 'rbf', gamma = gamma)

#fit/training set
r_sv1, sv1, y_predict1, score1, conf_int1, dist_hp1, model1 = \
    svm1.run_svm(train_1[cont._test_train__x_columns], train_1['1_vs_all'], C=C, random_state = 229)

train_conf1, train_per_conf1, train_limits1, train_confidence1 =
svm1.get_confusions(train_1['1_vs_all'], y_predict1)

#histogram
hist_dist1 = svm1.get_hist(dist_hp1, train_1['1_vs_all'], it = 1.1)

time_SVM1['Train'] = [time.time()-a]
a = time.time()

#test set
test_predict1, test_score1, test_conf_int1, test_dist_hp1 = \
    svm1.run_svm(test_1[cont._test_train__x_columns], test_1['1_vs_all'], C=C, random_state = 229,
        train = 'no', model = model1)

test_conf1, test_per_conf1, test_limits1, test_confidence1 = svm1.get_confusions(test_1['1_vs_all'],
test_predict1, conf_desired = .95)

#histogram
test_hist_dist1 = svm1.get_hist(test_dist_hp1, test_1['1_vs_all'], it = 1.2)

time_SVM1['Test'] = [time.time()-a]
##### SVM 2 vs All (2 is high wattage) #####
a = time.time()

#set Kernel
gamma = .075
C = 20
svm2 = SVM()
svm2.set_kernel(kernel = 'rbf', gamma = gamma)

#fit
n_sv2, sv2, y_predict2, score2, conf_int2, dist_hp2, model2 = \
```

```
svm2.run_svm(train_2[cont._test_train__x_columns], train_2['2_vs_all'], C=C, random_state = 229)

train_conf2, train_per_conf2, train_limits2, train_confidence2 =
svm2.get_confusions(train_2['2_vs_all'], y_predict2)

#histogram
hist_dist2 = svm2.get_hist(dist_hp2, train_2['2_vs_all'], it = 2.1)

time_SVM2['Train'] = [time.time()-a]
a = time.time()

#test set
test_predict2, test_score2, test_conf_int2, test_dist_hp2 = \
    svm2.run_svm(test_2[cont._test_train__x_columns], test_2['2_vs_all'], C=C, random_state = 229,
        train = 'no', model = model2)

test_conf2, test_per_conf2, test_limits2, test_confidence2 = svm2.get_confusions(test_2['2_vs_all'],
test_predict2, conf_desired = .95)

#histogram
test_hist_dist2 = svm2.get_hist(test_dist_hp2, test_2['2_vs_all'], it = 2.2)

time_SVM2['Test'] = [time.time()-a]
#####
#4 SVM's, 3 Confusion Matrixes/sets of histograms
#####

"""
Make a train and test table
run each svm's distances through each svms cdf
put them next to each other
pick the best one
"""

#Get all of your predictions from full training set and full test set
#SVM 0
y_predict0, score0, conf_int0, dist_hp0= \
    svm0.run_svm(train[cont._test_train__x_columns], train['0_vs_all'], C=C, random_state = 229,
        train = 'no', model = model0)

test_predict0, test_score0, test_conf_int0, test_dist_hp0 = \
    svm0.run_svm(test[cont._test_train__x_columns], test['0_vs_all'], C=C, random_state = 229,
        train = 'no', model = model0)
```

#### #SVM 1

```
y_predict1, score1, conf_int1, dist_hp1 = \
    svm1.run_svm(train[cont._test_train__x_columns], train['1_vs_all'], C=C, random_state = 229,
        train = 'no', model = model1)
```

```
test_predict1, test_score1, test_conf_int1, test_dist_hp1 = \
    svm1.run_svm(test[cont._test_train__x_columns], test['1_vs_all'], C=C, random_state = 229,
        train = 'no', model = model1)
```

#### #SVM 2

```
y_predict2, score2, conf_int2, dist_hp2 = \
    svm2.run_svm(train[cont._test_train__x_columns], train['2_vs_all'], C=C, random_state = 229,
        train = 'no', model = model2)
```

```
test_predict2, test_score2, test_conf_int2, test_dist_hp2 = \
    svm2.run_svm(test[cont._test_train__x_columns], test['2_vs_all'], C=C, random_state = 229,
        train = 'no', model = model2)
```

#### #split up the groups

```
train_class = pd.DataFrame(np.zeros((train.shape[0],4)), columns = ['SVM0', 'SVM1', 'SVM2',
    'Prediction'])
train_class['SVM0'] = hist_dist0.cdf(dist_hp0)
train_class['SVM1'] = hist_dist1.cdf(dist_hp1)
train_class['SVM2'] = hist_dist2.cdf(dist_hp2)
```

```
train_class['Prediction'] = train_class.idxmax(axis = 1)
train_class['Prediction'].replace(to_replace = class_to_num, inplace = True)
```

```
test_class = pd.DataFrame(np.zeros((test.shape[0],4)), columns = ['SVM0', 'SVM1', 'SVM2', 'Prediction'])
test_class['SVM0'] = hist_dist0.cdf(test_dist_hp0)
test_class['SVM1'] = hist_dist1.cdf(test_dist_hp1)
test_class['SVM2'] = hist_dist2.cdf(test_dist_hp2)
```

```
test_class['Prediction'] = test_class.idxmax(axis = 1)
test_class['Prediction'].replace(to_replace = class_to_num, inplace = True)
```

#### #Get confusion Matrix

```
final_test_conf, final_test_per_conf, final_test_limits, final_test_confidence =
    svm2.get_confusions(test['Class'], test_class['Prediction'], conf_desired = .95)
final_train_conf, final_train_per_conf, final_train_limits, final_train_confidence =
    svm2.get_confusions(train['Class'], train_class['Prediction'], conf_desired = .95)
```