Classification with Decision Trees and SVMs

Alperen Bitirgen

1 Part 1: Decision Tree

In this part of the assignment we had to set up a decision tree algorithm. Although I encountered some difficulties because I did not have much command of the Python class structure, I developed it in a way that could classify as a result. In the following sub-headings, I have included the tree structures created by the algorithm I obtained in cases of applying different constraints and their test accuracy values.

1.1 Information Gain

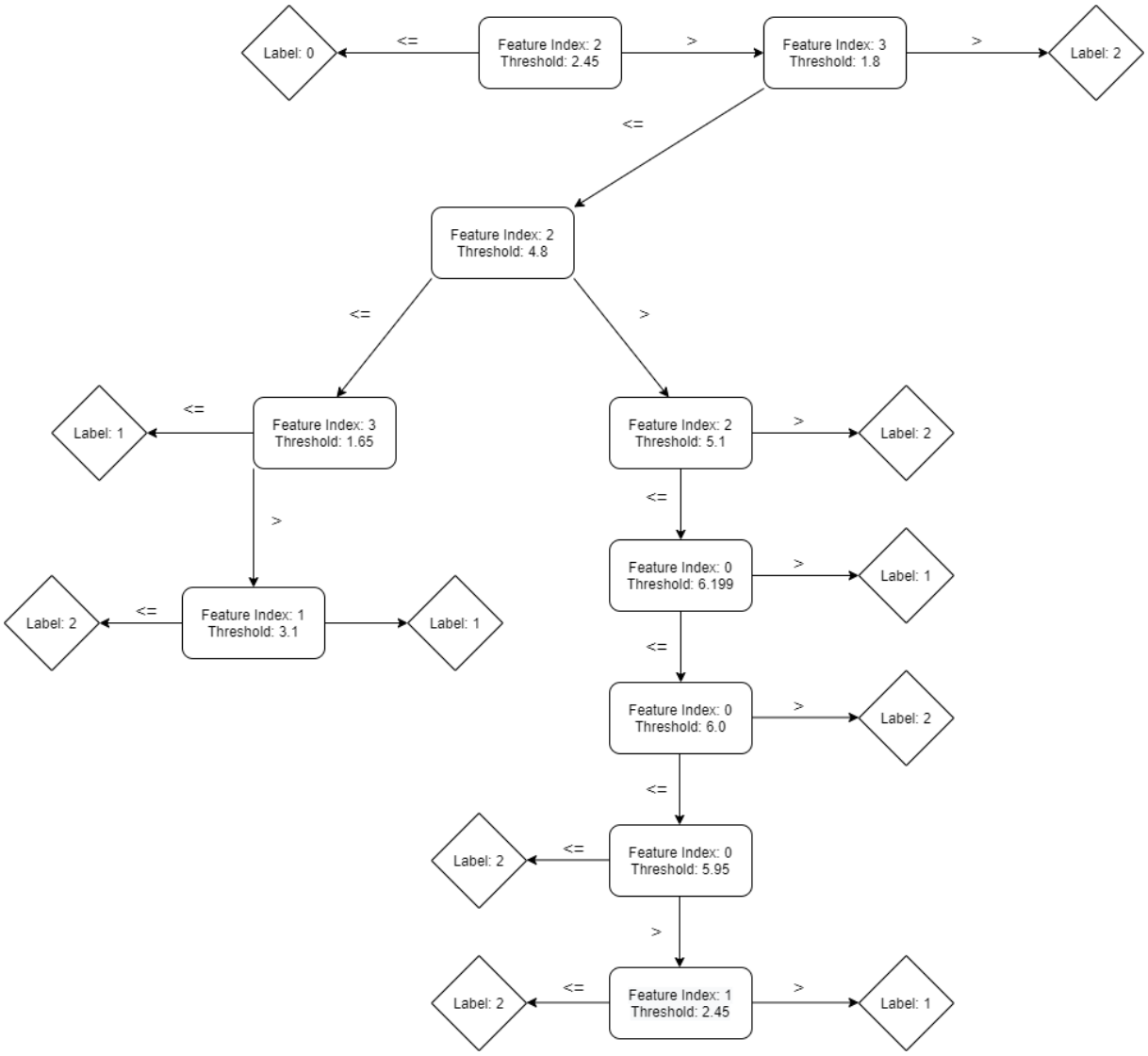


Figure 1: The representative tree diagram of using Information Gain while constructing the tree.

is attached with the name info_gain.png in the file I submitted. Out of 30 test data point, 28 of them labelled correctly. This gives us a test accuracy of **0.9333333333333333**.

1.2 Average Gini Index

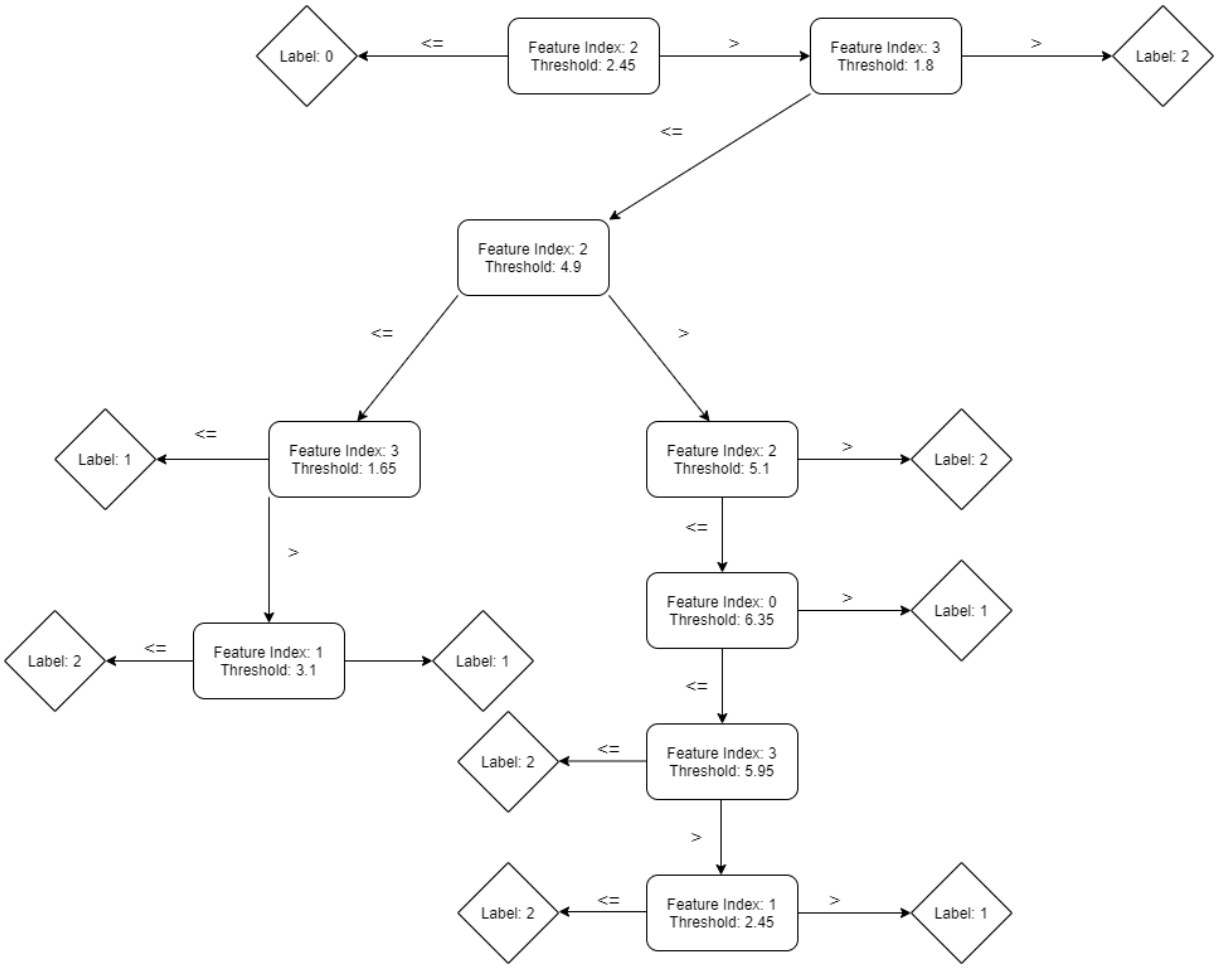


Figure 2: The representative tree diagram of using Average Gini Index while constructing the tree.

The representative Tree Diagram is attached with the name avg_gini.png in the file I submitted.Out of 30 test data point, 29 of them labelled correctly. This gives us a test accuracy of **0.9666666666666667**.

1.3 Information Gain with Chi-squared Pre-pruning

→ Here, I used pre-pruning with 75% confidence level. Out of 30 test data point, 28 of them labelled correctly. This gives us a test accuracy of **0.9333333333333333**.

→ Here, I used pre-pruning with 90% confidence level. The representative Tree Diagram is attached with the name preprun90_info_gain.png in the file I submitted. Out of 30 test data point, 28 of them labelled correctly. This gives us a test accuracy of **0.9333333333333333**.

→ Here, I used pre-pruning with 95% confidence level. Out of 30 test data point, 28 of them labelled correctly. This gives us a test accuracy of **0.9333333333333333**.

1.4 Average Gini Index with Chi-squared Pre-pruning

→ Here, I used pre-pruning with 75% confidence level. Out of 30 test data point, 30 of them labelled correctly. This gives us a test accuracy of **1.0**.

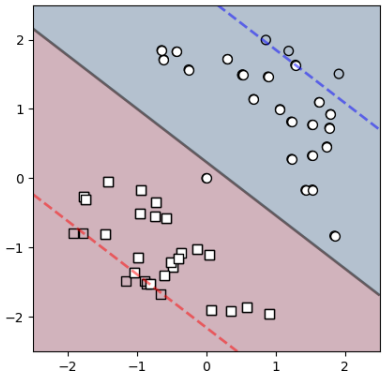
→ Here, I used pre-pruning with 90% confidence level. The representative Tree Diagram is attached with the name preprun90_avg_gini.png in the file I submitted.Out of 30 test data point, 29 of them labelled correctly. This gives us a test accuracy of **0.9666666666666667**.

→ Here, I used pre-pruning with 95% confidence level. Out of 30 test data point, 30 of them labelled correctly. This gives us a test accuracy of **1.0**.

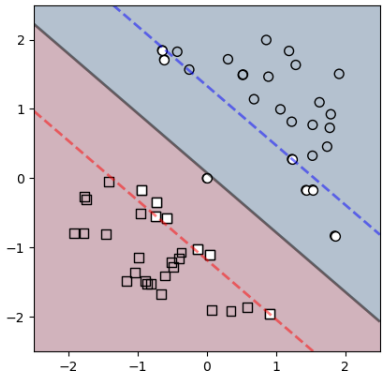
2 Part 2: Support Vector Machine

2.1 First Part

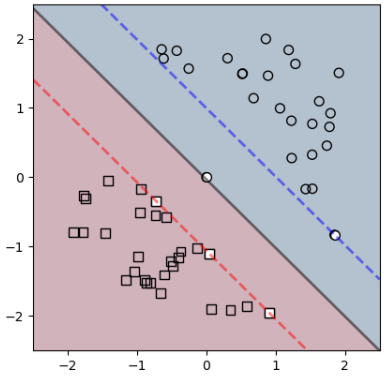
In this section, the effect of the change on the C value on the classifier was examined using only the linear kernel. I have included the figures I obtained as a result of the code I wrote below. As can be seen from the figures, increasing the C value narrowed the hyper-plane margin, resulting in a decrease in the number of incorrectly classified data points. Although this change persisted visibly up to C = 10, it did not cause a discernible change between C = 100 and C = 10.



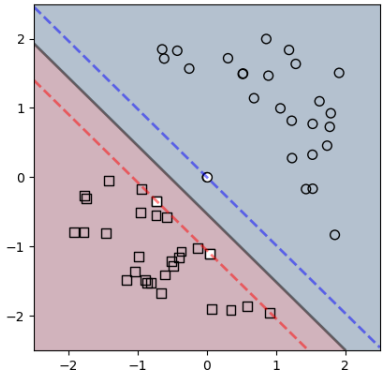
(a) Linear Kernel with C = 0.01



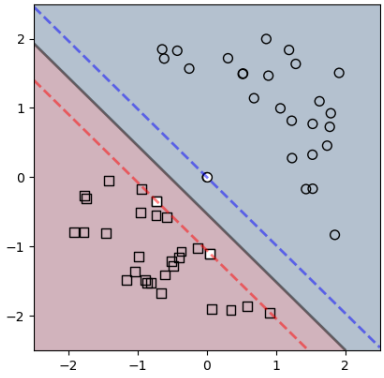
(b) Linear Kernel with C = 0.1



(c) Linear Kernel with C = 1.0



(d) Linear Kernel with C = 10.0

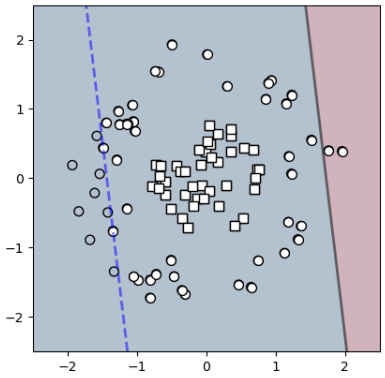


(e) Linear Kernel with C = 100.0

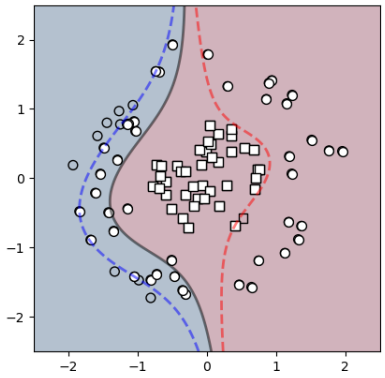
Figure 3: Effect of parameter C while using Linear Kernel.

2.2 Second Part

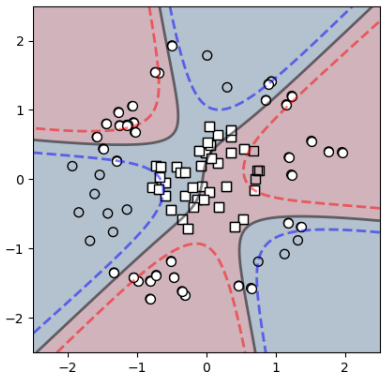
In this section, using different kernels and keeping the C value constant, only the effect of kernel selection on the classifier is examined. I have given the figures below as a result of the code I wrote. As can be seen from the figures, changing the kernel while C = 1 allowed us to obtain different hyper-plane margins. This is change. As misclassification cases were a corollary of these changing margins, changes were also seen in the number of misclassified data points and the points where misclassified points clustered. When the figures are examined, it is obvious that the Radial Basis Function (rbf) gives the best results.



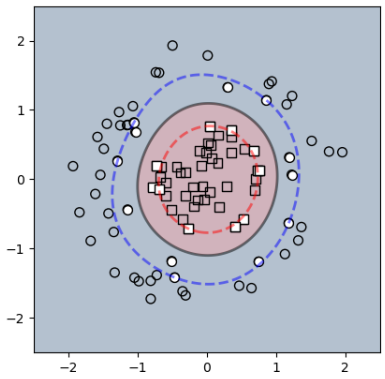
(a) Linear Kernel with C = 1



(b) Polynomial Kernel with C = 1



(c) Sigmoid Kernel with C = 1



(d) Radial Basis Function Kernel with C = 1

Figure 4: Effect of the kernel type.

2.3 Third Part

In this part, the classifier effect of these variables on the balanced data set was examined by using different kernels and C values. I have given the accuracy values that I obtained as a result of the code I wrote in the tables below. As a result of the changes made on different hyper-parameters, I found that the optimization that gave the best cross validation result was obtained with **C = 10.0 and Gamma = 0.1** values over the **rbf kernel**. And I calculated the accuracy of these hyper-parameters on the test-set as **0.884**.

gamma	C				
	0.01	0.1	1	10	100
-	0.762667	0.803333	0.802000	0.756000	0.730000

Table 1: Linear kernel

gamma	C				
	0.01	0.1	1	10	100
0.00001	0.510000	0.510000	0.510000	0.510000	0.735333
0.0001	0.510000	0.510000	0.510000	0.735333	0.774667
0.001	0.510000	0.510000	0.734000	0.792000	0.847333
0.01	0.510000	0.737333	0.822667	0.884000	0.877333
0.1	0.510000	0.510667	0.896667	0.901333	0.901333
1	0.510000	0.510000	0.510000	0.510000	0.510000

Table 2: RBF kernel

gamma	C				
	0.01	0.1	1	10	100
0.00001	0.510000	0.510000	0.510000	0.510000	0.706000
0.0001	0.510000	0.510000	0.510000	0.706000	0.762667
0.001	0.510000	0.510000	0.703333	0.762667	0.799333
0.01	0.510000	0.510000	0.548000	0.453333	0.428000
0.1	0.510000	0.510000	0.510000	0.510000	0.510667
1	0.510000	0.510000	0.510000	0.510000	0.510000

Table 3: Sigmoid kernel

		C				
degree	gamma	0.01	0.1	1	10	100
2	0.00001	0.510000	0.510000	0.510000	0.510000	0.510000
	0.0001	0.510000	0.510000	0.510000	0.510000	0.510000
	0.001	0.510000	0.510000	0.510000	0.731333	0.793333
	0.01	0.510000	0.731333	0.793333	0.873333	0.859333
	0.1	0.793333	0.873333	0.859333	0.858667	0.858667
	1.0	0.859333	0.858667	0.858667	0.858667	0.858667
3	0.00001	0.510000	0.510000	0.510000	0.510000	0.510000
	0.0001	0.510000	0.510000	0.510000	0.510000	0.510000
	0.001	0.510000	0.510000	0.510000	0.562000	0.741333
	0.01	0.562000	0.741333	0.812667	0.875333	0.867333
	0.1	0.875333	0.867333	0.867333	0.867333	0.867333
	1.0	0.867333	0.867333	0.867333	0.867333	0.867333
4	0.00001	0.510000	0.510000	0.510000	0.510000	0.510000
	0.0001	0.510000	0.510000	0.510000	0.510000	0.510000
	0.001	0.510000	0.510000	0.510000	0.510000	0.624000
	0.01	0.624000	0.746667	0.815333	0.874000	0.879333
	0.1	0.879333	0.876000	0.876000	0.876000	0.876000
	1.0	0.876000	0.876000	0.876000	0.876000	0.876000
5	0.00001	0.510000	0.510000	0.510000	0.510000	0.510000
	0.0001	0.510000	0.510000	0.510000	0.510000	0.510000
	0.001	0.510000	0.510000	0.510000	0.510000	0.510000
	0.01	0.642667	0.744000	0.808667	0.869333	0.873333
	0.1	0.870667	0.870667	0.870667	0.870667	0.870667
	1.0	0.870667	0.870667	0.870667	0.870667	0.870667
6	0.00001	0.510000	0.510000	0.510000	0.510000	0.510000
	0.0001	0.510000	0.510000	0.510000	0.510000	0.510000
	0.001	0.510000	0.510000	0.510000	0.510000	0.510000
	0.01	0.640000	0.738667	0.804667	0.864000	0.861333
	0.1	0.854667	0.854667	0.854667	0.854667	0.854667
	1.0	0.854667	0.854667	0.854667	0.854667	0.854667

Table 4: Polynomial kernel

2.4 Fourth part

2.4.1 Without handling the imbalance problem

Accuracy is not a good performance measure in this context, as false positives are considered to have costs equal to false negatives in imbalanced data-sets, but often pretending costs are equal leads to critical false inferences. We can clearly observe this in the data set we have. In fact, only 3 of the 79 data points that should have a 0 label are classified as 0. We can also notice this situation when we examine the specificity value, because the very low value of this value is the most obvious indicator of the incorrect classification of the 0 class.

		Actual Label	
		1	0
Predicted Label	1	1421	76
	0	0	3

Table 5: Confusion Matrix before handling the imbalance problem

→ **Precision: 0.9492317969271877**
→ **Recall: 1.0**
→ **F1-score: 0.973954763536669**
→ **Specificity: 0.0379746835443038**
→ **Test Accuracy: 0.9493333333333334**

2.4.2 Oversampling the minority class

Over-sampling the minority class, where it is class label 0 in this case, obviously increases the specificity. It is able to classify better the actual 0 class labeled data points as 0. It also increases the test accuracy.

→ **Precision: 0.9669876203576341**

		Actual Label	
		1	0
Predicted Label	1	1406	48
	0	15	31

Table 6: Confusion Matrix with over-sampling the minority class

→ **Recall: 0.9894440534834623**
→ **F1-score: 0.978086956521739**
→ **Specificity: 0.3924050632911392**
→ **Test Accuracy: 0.958**

2.4.3 Undersampling the majority class

When we under-sampled the majority class, the specificity value became closer to the recall value. However, this brings us a steep decrease in the test accuracy. Initially, the accuracy value of 0.95, which we obtained without any processing on the imbalance data-set, dropped to 0.79 in this case.

→ **Precision: 0.9767040552200172**
→ **Recall: 0.796622097114708**
→ **F1-score: 0.8775193798449613**

		Actual Label	
		1	0
Predicted Label	1	1132	27
	0	289	52

Table 7: Confusion Matrix with under-sampling the majority class

→ **Specificity: 0.6582278481012658**
→ **Test Accuracy: 0.7893333333333333**

2.4.4 Setting the class_weight to balanced

I can get the best test accuracy at this stage. It has given an even better accuracy result than the oversampling of the minority class, which has already given the highest accuracy before. Our precision, recall and f1-score values also gave higher values compared to the over-sampling the minority class operation.

		Actual Label	
		1	0
Predicted Label	1	1401	42
	0	20	37

Table 8: Confusion Matrix with balanced class weight

→ **Precision: 0.9708939708939709**
→ **Recall: 0.9859254046446164**
→ **F1-score: 0.9783519553072626**
→ **Specificity: 0.46835443037974683**
→ **Test Accuracy: 0.9586666666666667**