Report

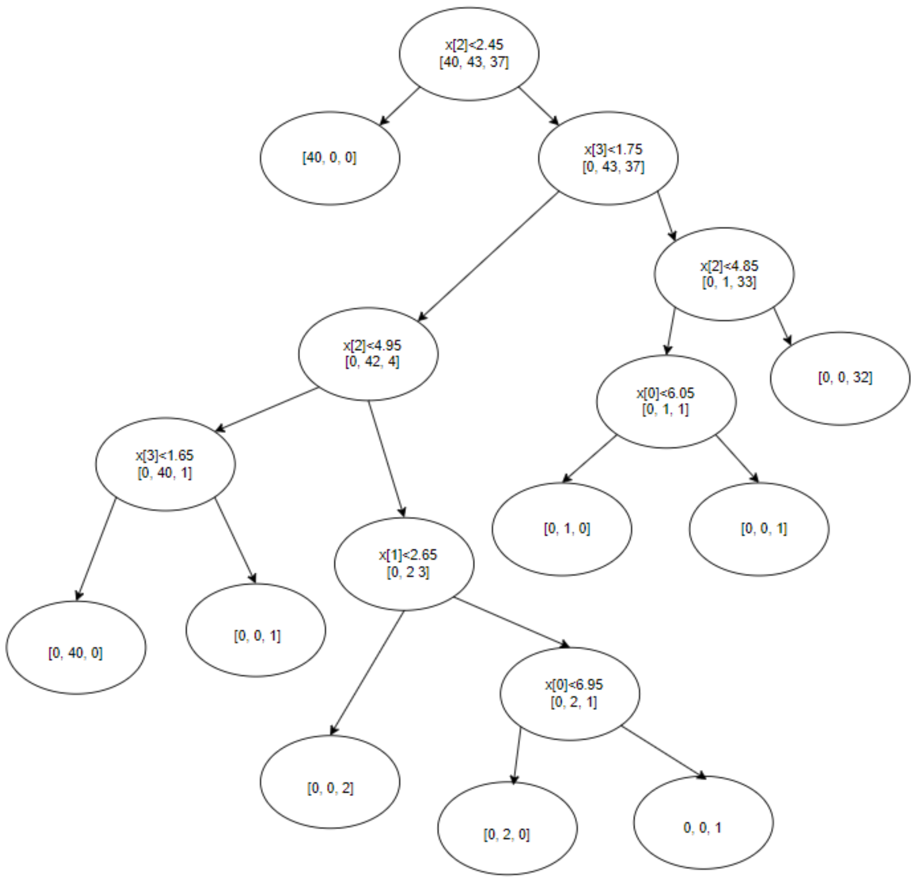
Ahmet Berker KOÇ
2232320

June 12, 2021

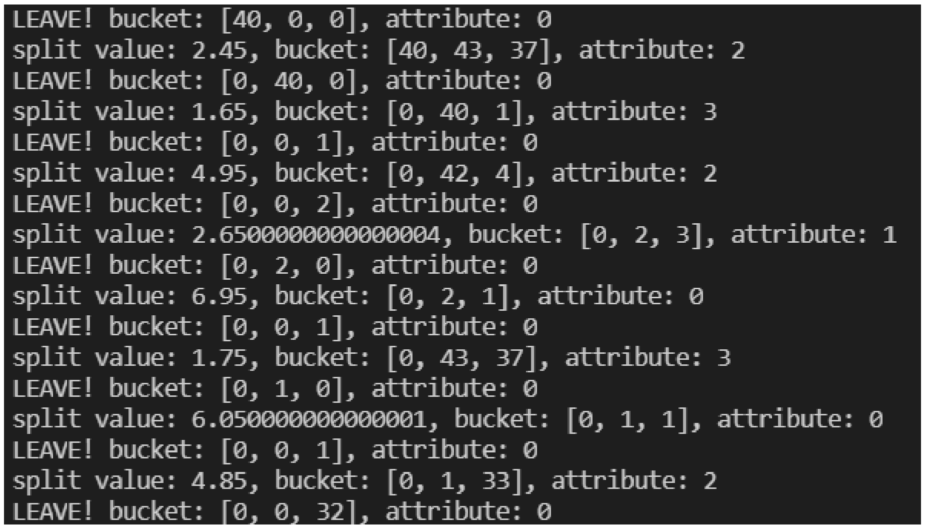
1 Part 1: Decision Tree

1.1 Information Gain

Here is my tree which created by using information gain without pruning process

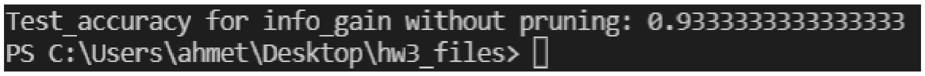


Here is my decision tree printing. Printing is done according to inorder traversal



```
LEAVE! bucket: [40, 0, 0], attribute: 0
split value: 2.45, bucket: [40, 43, 37], attribute: 2
LEAVE! bucket: [0, 40, 0], attribute: 0
split value: 1.65, bucket: [0, 40, 1], attribute: 3
LEAVE! bucket: [0, 0, 1], attribute: 0
split value: 4.95, bucket: [0, 42, 4], attribute: 2
LEAVE! bucket: [0, 0, 2], attribute: 0
split value: 2.6500000000000004, bucket: [0, 2, 3], attribute: 1
LEAVE! bucket: [0, 2, 0], attribute: 0
split value: 6.95, bucket: [0, 2, 1], attribute: 0
LEAVE! bucket: [0, 0, 1], attribute: 0
split value: 1.75, bucket: [0, 43, 37], attribute: 3
LEAVE! bucket: [0, 1, 0], attribute: 0
split value: 6.050000000000001, bucket: [0, 1, 1], attribute: 0
LEAVE! bucket: [0, 0, 1], attribute: 0
split value: 4.85, bucket: [0, 1, 33], attribute: 2
LEAVE! bucket: [0, 0, 32], attribute: 0
```

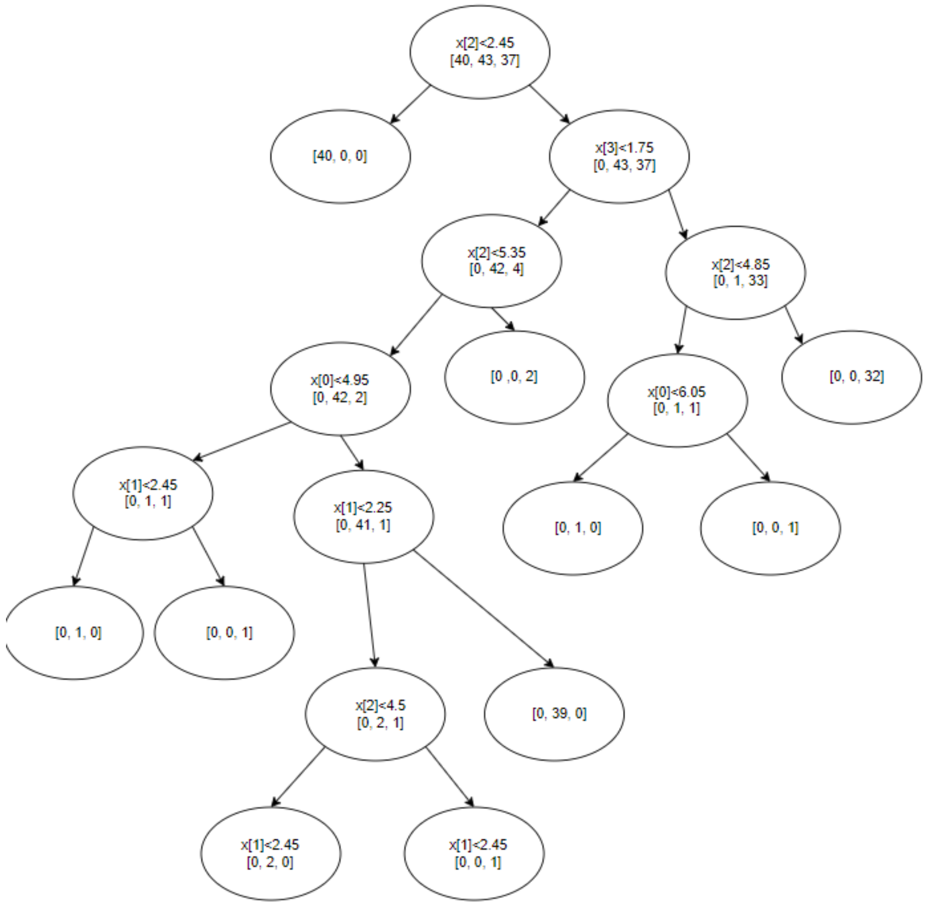
Here is test result (my accuracy for test data)



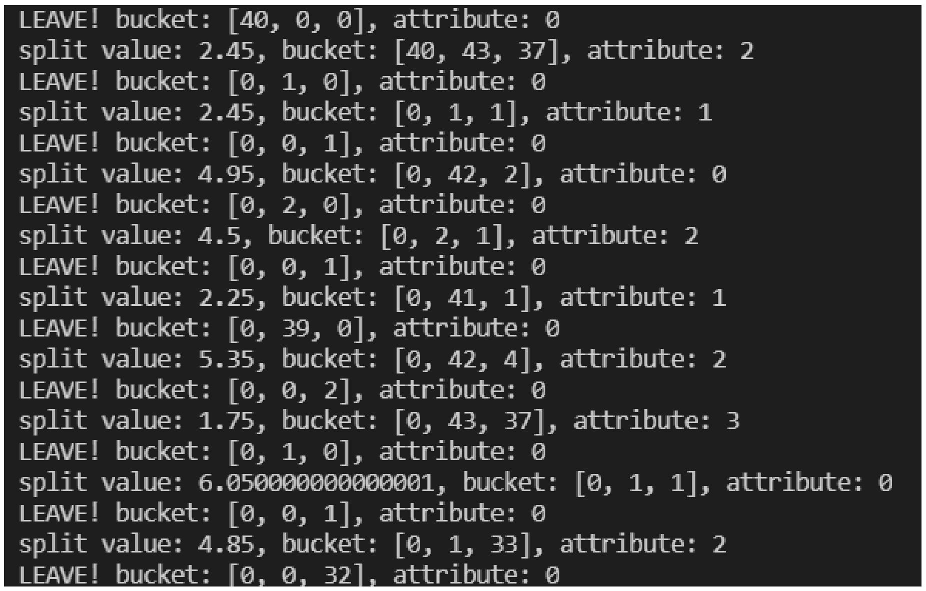
```
Test_accuracy for info_gain without pruning: 0.9333333333333333
PS C:\Users\ahmet\Desktop\hw3_files>
```

1.2 Average Gini Index

Here is my tree which created by using average gini index without pruning process

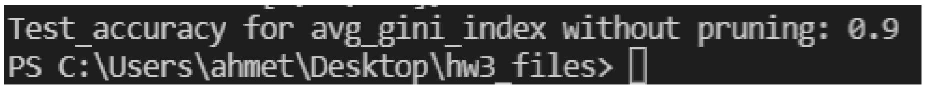


Here is my decision tree printing. Printing is done according to inorder traversal



```
LEAVE! bucket: [40, 0, 0], attribute: 0
split value: 2.45, bucket: [40, 43, 37], attribute: 2
LEAVE! bucket: [0, 1, 0], attribute: 0
split value: 2.45, bucket: [0, 1, 1], attribute: 1
LEAVE! bucket: [0, 0, 1], attribute: 0
split value: 4.95, bucket: [0, 42, 2], attribute: 0
LEAVE! bucket: [0, 2, 0], attribute: 0
split value: 4.5, bucket: [0, 2, 1], attribute: 2
LEAVE! bucket: [0, 0, 1], attribute: 0
split value: 2.25, bucket: [0, 41, 1], attribute: 1
LEAVE! bucket: [0, 39, 0], attribute: 0
split value: 5.35, bucket: [0, 42, 4], attribute: 2
LEAVE! bucket: [0, 0, 2], attribute: 0
split value: 1.75, bucket: [0, 43, 37], attribute: 3
LEAVE! bucket: [0, 1, 0], attribute: 0
split value: 6.050000000000001, bucket: [0, 1, 1], attribute: 0
LEAVE! bucket: [0, 0, 1], attribute: 0
split value: 4.85, bucket: [0, 1, 33], attribute: 2
LEAVE! bucket: [0, 0, 32], attribute: 0
```

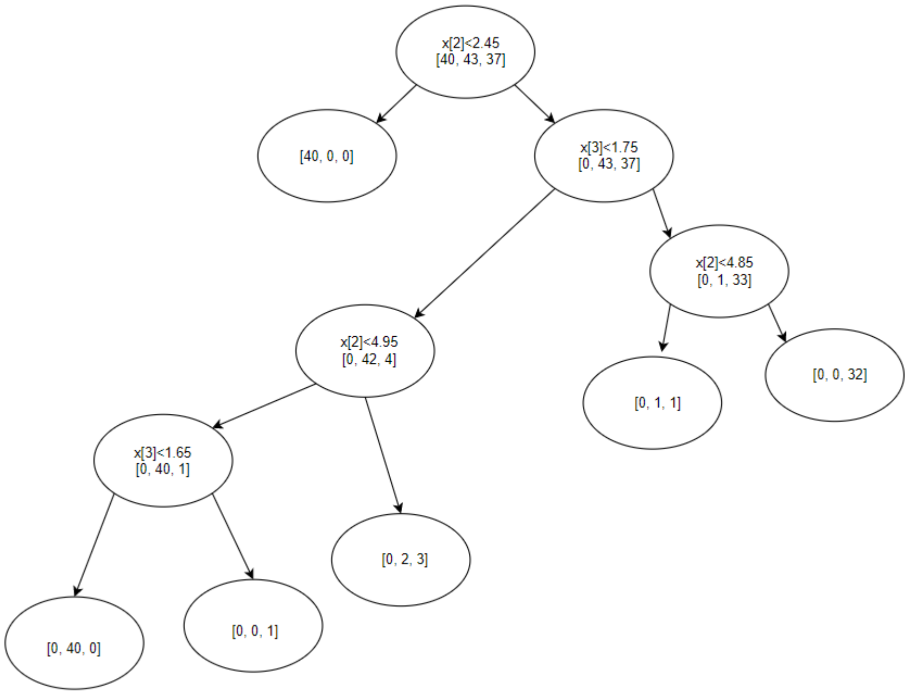
Here is test result (my accuracy for test data)



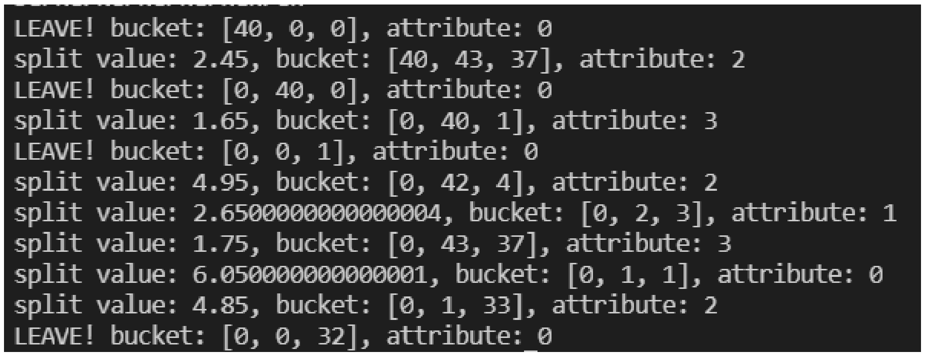
```
Test_accuracy for avg_gini_index without pruning: 0.9
PS C:\Users\ahmet\Desktop\hw3_files> 
```

1.3 Information Gain with Chi-squared Pre-pruning

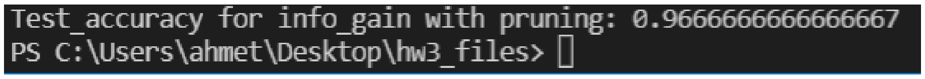
Here is my tree which created by using information gain index with pruning process



Here is my decision tree printing. Printing is done according to inorder traversal

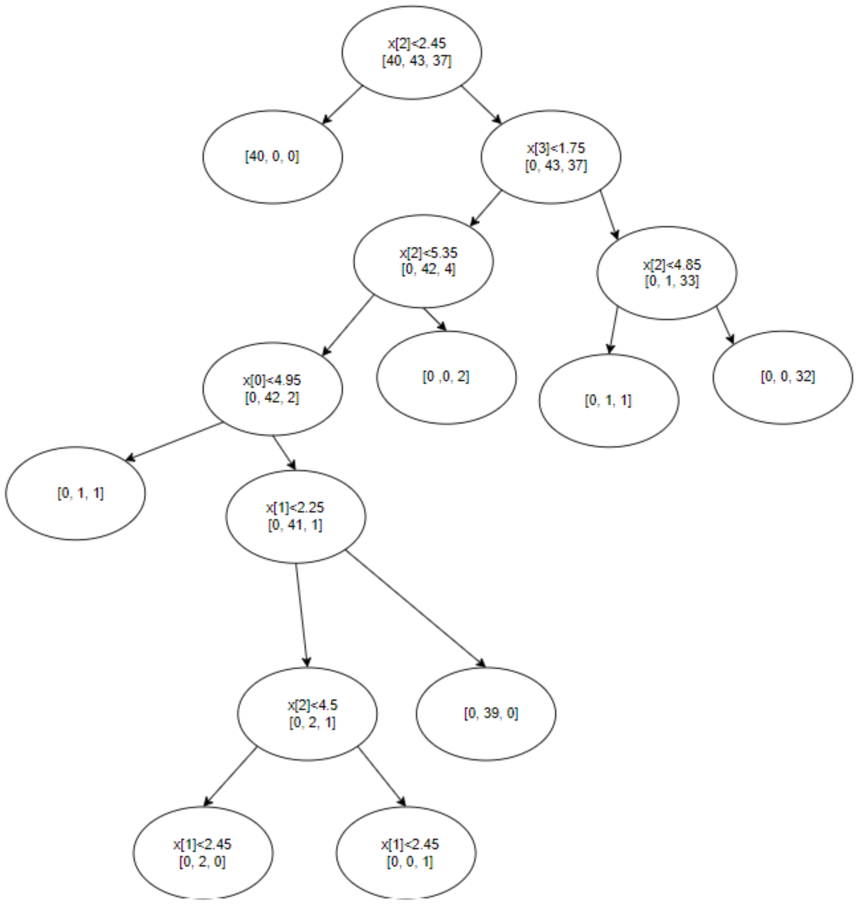

```
LEAVE! bucket: [40, 0, 0], attribute: 0
split value: 2.45, bucket: [40, 43, 37], attribute: 2
LEAVE! bucket: [0, 40, 0], attribute: 0
split value: 1.65, bucket: [0, 40, 1], attribute: 3
LEAVE! bucket: [0, 0, 1], attribute: 0
split value: 4.95, bucket: [0, 42, 4], attribute: 2
split value: 2.6500000000000004, bucket: [0, 2, 3], attribute: 1
split value: 1.75, bucket: [0, 43, 37], attribute: 3
split value: 6.050000000000001, bucket: [0, 1, 1], attribute: 0
split value: 4.85, bucket: [0, 1, 33], attribute: 2
LEAVE! bucket: [0, 0, 32], attribute: 0
```

Here is test result (my accuracy for test data)

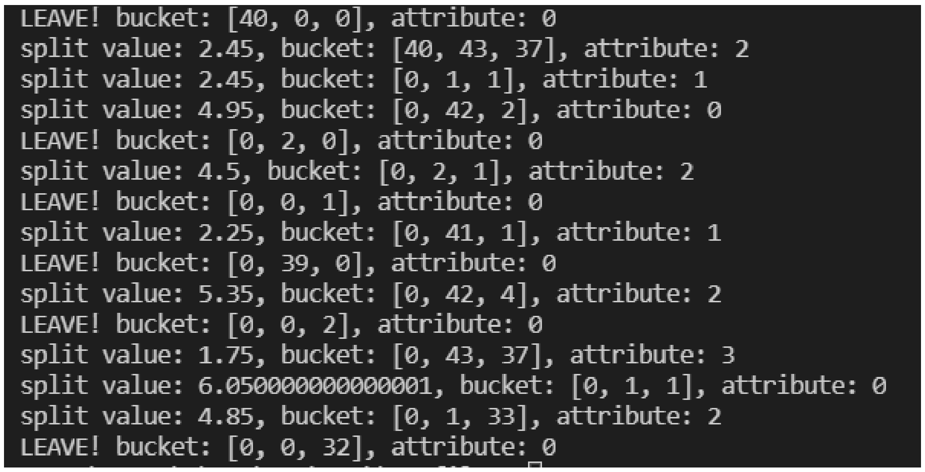

```
Test_accuracy for info_gain with pruning: 0.9666666666666667
PS C:\Users\ahmet\Desktop\hw3_files> 
```

1.4 Average Gini Index with Chi-squared Pre-pruning

Here is my tree which created by using average gini index index with pruning process

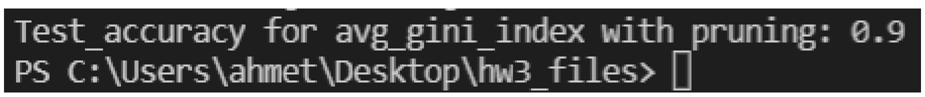


Here is my decision tree printing. Printing is done according to inorder traversal



```
LEAVE! bucket: [40, 0, 0], attribute: 0
split value: 2.45, bucket: [40, 43, 37], attribute: 2
split value: 2.45, bucket: [0, 1, 1], attribute: 1
split value: 4.95, bucket: [0, 42, 2], attribute: 0
LEAVE! bucket: [0, 2, 0], attribute: 0
split value: 4.5, bucket: [0, 2, 1], attribute: 2
LEAVE! bucket: [0, 0, 1], attribute: 0
split value: 2.25, bucket: [0, 41, 1], attribute: 1
LEAVE! bucket: [0, 39, 0], attribute: 0
split value: 5.35, bucket: [0, 42, 4], attribute: 2
LEAVE! bucket: [0, 0, 2], attribute: 0
split value: 1.75, bucket: [0, 43, 37], attribute: 3
split value: 6.050000000000001, bucket: [0, 1, 1], attribute: 0
split value: 4.85, bucket: [0, 1, 33], attribute: 2
LEAVE! bucket: [0, 0, 32], attribute: 0
```

Here is test result (my accuracy for test data)



```
Test_accuracy for avg_gini_index with pruning: 0.9
PS C:\Users\ahmet\Desktop\hw3_files>
```

Comment:

The accuracy are almost same for gini index and information gain. Information gain accuracy is higher than that of gini index. After pruning, results have not change much. In fact, before and after pruning result for gini index remain same. In addition for information gain accuracy increase with 0.033. This depends on the data distribution in the data-set.

2 Part 2: Support Vector Machine

2.1 First Part

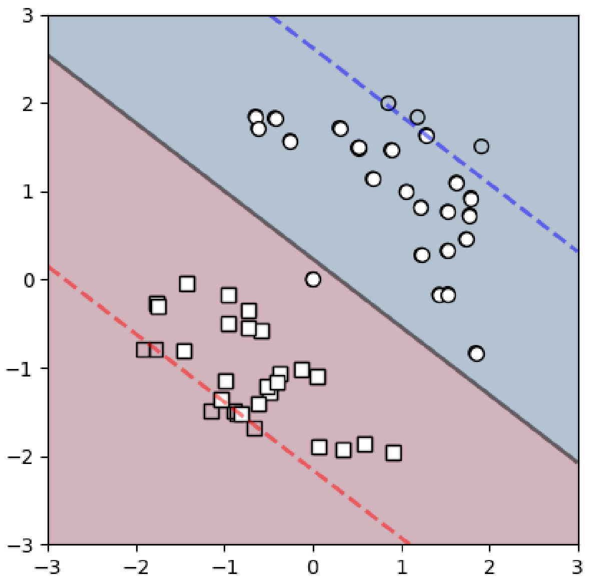


Figure 1: Result for C=0.01

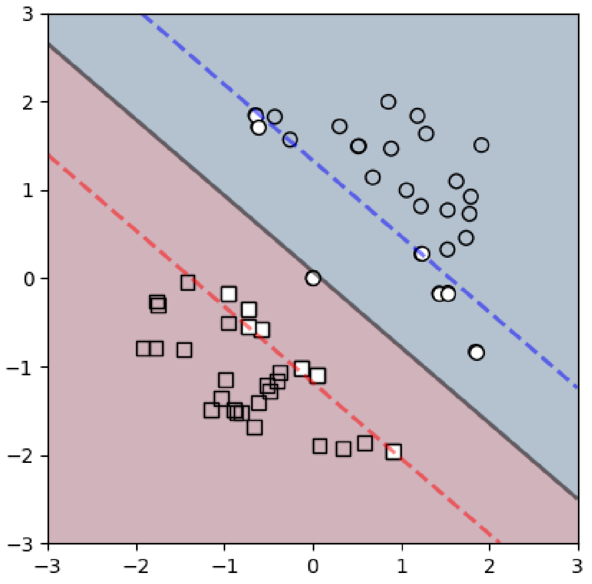


Figure 2: Result for C=0.1

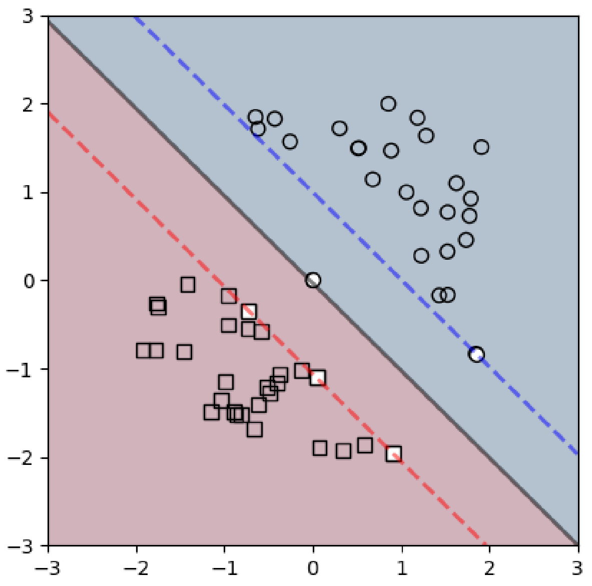


Figure 3: Result for C=1

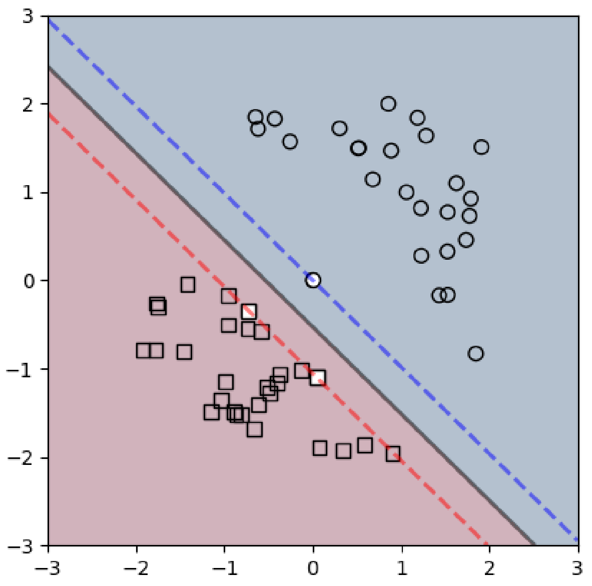


Figure 4: Result for C=10

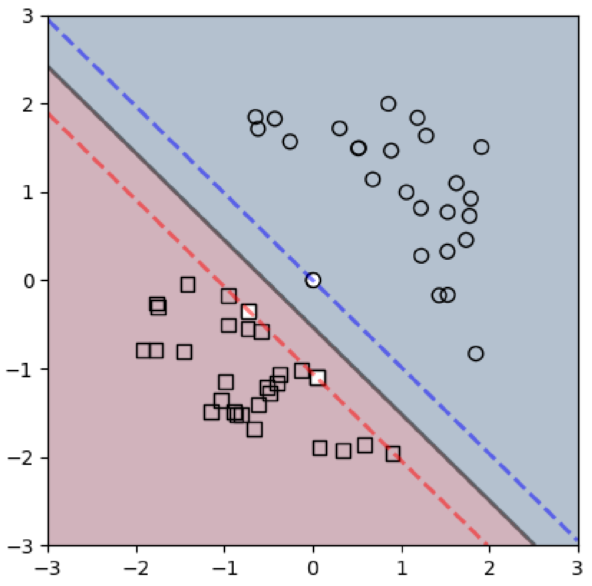


Figure 5: Result for C=100

Comment:
When C increase, misclassification decreases. If C become infinity, there would be no misclassification which means hard margin. In addition, when C increase, margin become narrow as you can see the figures. When C=100 and C=10, results are similar. This means this C values gave us a sufficient results. Decreasing in misclassification also means that models are getting more and more complex.
In soft margin classification, model tries to minimize this formula

$$\frac{1}{2}w.w + C\sum_{k=1}^{R}\xi_k$$

When C (regularization parameter) is small, minimization process became easy; however, this causes that ξ value minimization doesn't happen well. Therefore, model have more misclassification

2.2 Second Part

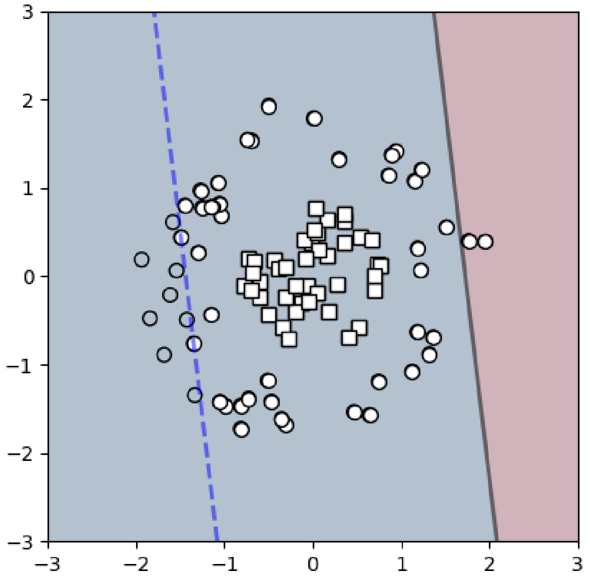


Figure 6: Result for C=1 and kernel values linear

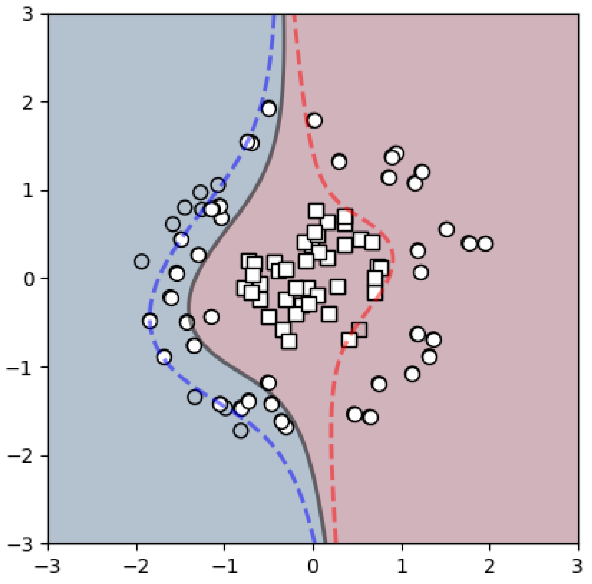


Figure 7: Result for C=1 and kernel values polynomial

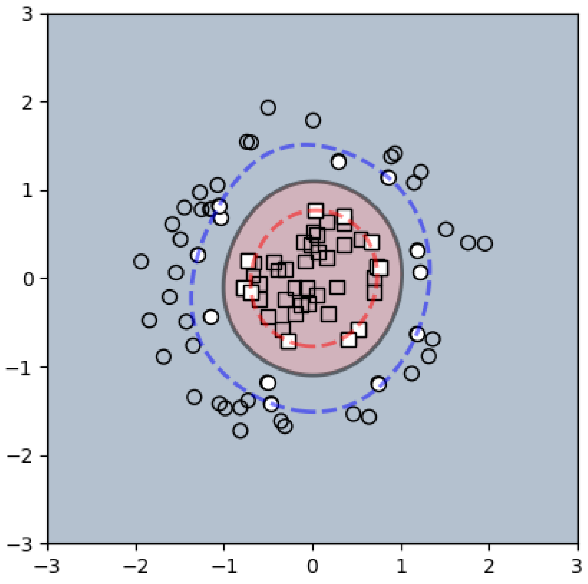


Figure 8: Result for C=1 and kernel values rbf

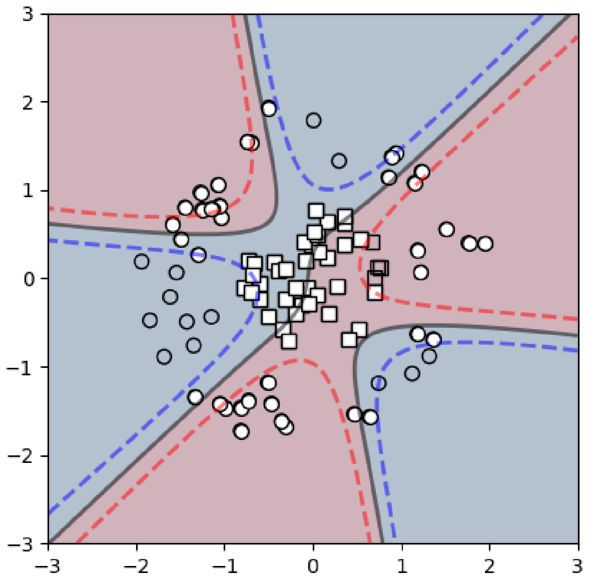


Figure 9: Result for C=1 and kernel values sigmoid

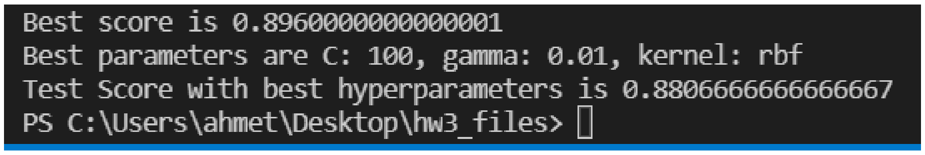
Comment:
The most appropriate choice is rbf for this data-set. Although C value is 1 and it enables misclassification. The best separation is done by rbf as you can see from the figures. Data-set has not a linearly separable data for 2D. Therefore, linear kernel is not suitable. Also, polynomial and sigmoid is not suitable for this data-set. However, rbf determines the hyper plane circularly. This enable it to separate data according to its distribution.

2.3 Third Part

For the best paremeters are as follows

- C=100
- gamma=0.01
- kernel=rbf

For the best model I obtain the accuracy as



```
Best score is 0.8960000000000001
Best parameters are C: 100, gamma: 0.01, kernel: rbf
Test Score with best hyperparameters is 0.8806666666666667
PS C:\Users\ahmet\Desktop\hw3_files> 
```

Best score is obtained by using clf.best_score_ function after fitting Test score with best hyper-parameters is obtained by using clf.score(test_data, test_label)

gamma	C				
	0.01	0.1	1	10	100
-	0.774	0.811	0.779	0.738	0.730

Table 1: Linear kernel

gamma	C				
	0.01	0.1	1	10	100
0.00001	0.510	0.510	0.510	0.511	0.747
0.0001	0.510	0.510	0.511	0.749	0.796
0.001	0.510	0.511	0.751	0.828	0.857
0.01	0.510	0.761	0.869	0.894	**0.896**
0.1	0.510	0.510	0.883	0.885	0.885
1	0.510	0.510	0.510	0.510	0.510

Table 2: RBF kernel

gamma	C				
	0.01	0.1	1	10	100
0.00001	0.510	0.510	0.510	0.510	0.510
0.0001	0.510	0.510	0.510	0.510	0.510
0.001	0.510	0.510	0.551	0.737	0.810
0.01	0.737	0.810	0.875	0.867	0.867
0.1	0.867	0.867	0.867	0.867	0.867
1	0.867	0.867	0.867	0.867	0.867

Table 3: Polynomial kernel

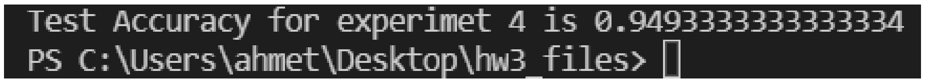
gamma	C				
	0.01	0.1	1	10	100
0.00001	0.510	0.510	0.510	0.510	0.737
0.0001	0.510	0.510	0.510	0.737	0.774
0.001	0.510	0.510	0.736	0.768	0.787
0.01	0.510	0.511	0.337	0.687	0.681
0.1	0.510	0.510	0.510	0.510	0.510
1	0.510	0.510	0.510	0.510	0.510

Table 4: Sigmoid kernel

2.4 Fourth part

2.4.1 Without handling the imbalance problem

Here is my test occuracy. It is very high



```
Test Accuracy for experimet 4 is 0.9493333333333334
PS C:\Users\ahmet\Desktop\hw3_files> 
```

Comment:
When it examined alone, accuracy is not a good performance metric because for unbalanced data-set majority classes can be very dominant and model may start to predict according to this majority class(for example all data is predicted as majority class). This causes that models doesn't learn the minority class and therefore model cannot separate it from majority class. As seen in the Figure 10, although model is bad at classification of majority and minority class. Accuracies are very high. Even, accuracy is increasing while model are getting worse. Therefore, accuracy is not enough alone to interpret the model's success

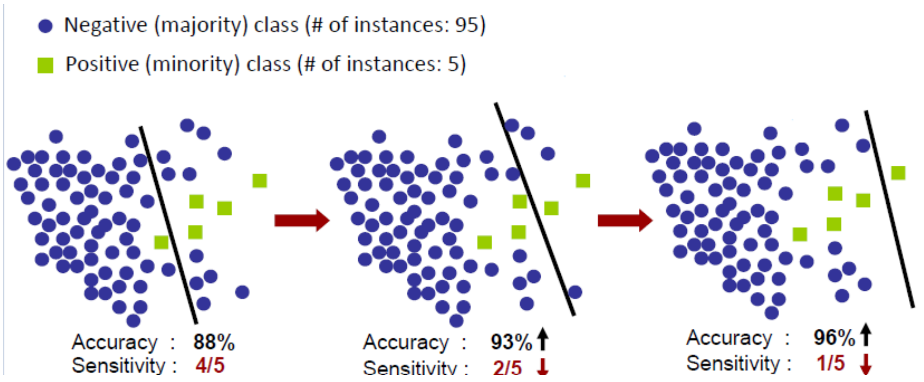
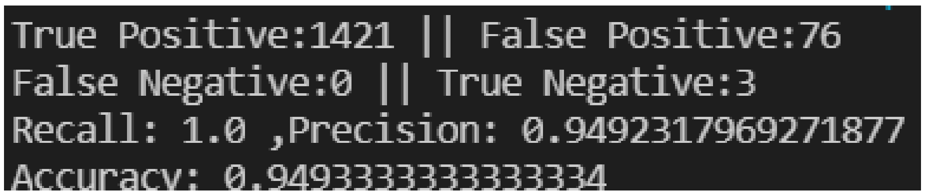

● Negative (majority) class (# of instances: 95)
🟩 Positive (minority) class (# of instances: 5)
Accuracy : 88%
Sensitivity : 4/5
Accuracy : 93%
Sensitivity : 2/5
Accuracy : 96%
Sensitivity : 1/5

Figure 10: From lecture notes unbalance data set effect on accuracy

Here is my confision matrix and performance metrics Recall,Precision and accuracy



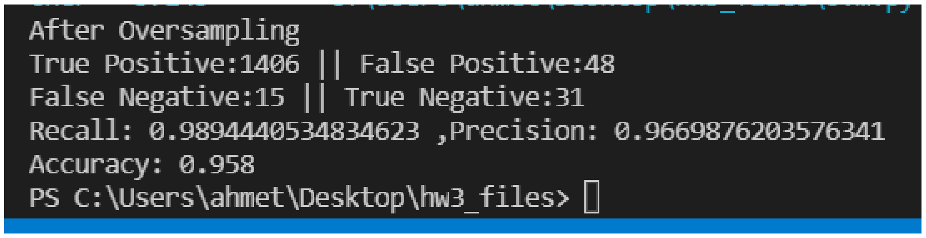
```
True Positive:1421 || False Positive:76
False Negative:0 || True Negative:3
Recall: 1.0 ,Precision: 0.9492317969271877
Accuracy: 0.9493333333333334
```

Comment:
We can say that 1 class of the data-set is too dominant. While examining the result, I see that almost all prediction is 1 ;therefore, there is no false negative

and this prediction leads to false positive. Also, there is small number of true negative. These values(TP, TN, FP, FN) cause the recall is 1 and also precision is very high because for both numerator is TP and TP is dominant. Similarly, accuracy is very high because TP is very dominant again.

2.4.2 Oversampling the minority class

Here is my test accuracy, confusion matrix



```
After Oversampling
True Positive:1406 || False Positive:48
False Negative:15 || True Negative:31
Recall: 0.9894440534834623 ,Precision: 0.9669876203576341
Accuracy: 0.958
PS C:\Users\ahmet\Desktop\hw3_files> 
```

In this part, I oversample minority class in the training data by copying the minority class data. I firstly find number of data for majority class (1) and minority class(0). Then I find the ratio. I repeat the ratio-1 times copying minority data. The ratio between majority and minorty class was 1379:121 before oversampling. After oversampling the ratio become 1379:1331

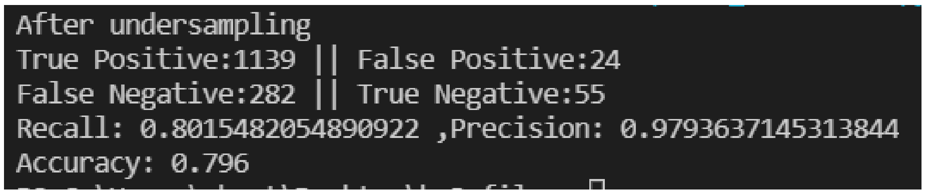
Different from first experiment, there is FN, TN increase and FP decrease. This means that our model make more prediction for 0 class. This also means while training, it learn the 0 class comparing with first example (unbalanced). Still it learn the positives well

Comparing with experiment1
Recall decrease because there are some false negatives anymore and true positive decrease Precision increase because false positives decrease. Accuracy increase because true negative increase more than increase in false negative. Also, decrease in false positives promote the accuracy

Main Disadvantage: Computation limitations and inefficient use of memory

2.4.3 Undersampling the majority class

Here is my test accuracy, confusion matrix



```
After undersampling
True Positive:1139 || False Positive:24
False Negative:282 || True Negative:55
Recall: 0.8015482054890922 ,Precision: 0.9793637145313844
Accuracy: 0.796
```

In this part, I undersample the majority class by deleting majority class data. I firstly find the difference between majority class (1) and minority class(0). Then I delete majority class until reach the same number of class. The ratio between majority and minority class was 1379:121 before oversampling. After undersampling the ratio become 121:121 (ratio 1)

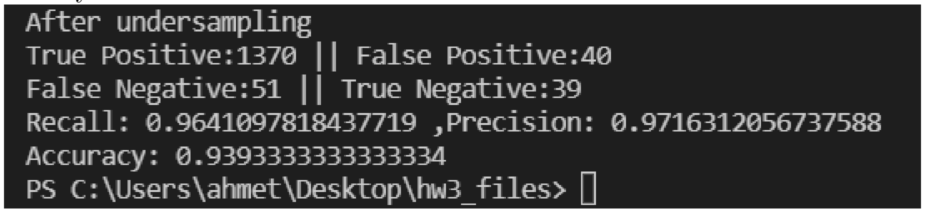
Different from first experiment, there are lots of FN, TN increase and FP decrease. This means that our model make more prediction for 0 class. This again means that while training, it learn the 0 class comparing with first example (unbalanced). Still it learn the positives well. Accuracy is still very high 0.985

Different from second experiment(oversample), there is more negative prediction. Thanks to this, thre is more TN and FN. Also, In this part, I set the ratio as 1 differently. This make more balanced data. In experiment 2, majority class have more data despite the oversample. However, while deleting, I may delete more informative data (I delete with data order, I don't try to choose more informative data). Therefore, ratio 1 and deleting process may leads to decrease in accuracy comparing with experiment 1 and 0

Comparing with experiment1
Recall decrease because there are a lot of false negatives anymore and true positive decrease. Precision increase because false positives decrease much. Accuracy decrease. Although true negative increase than increase in false negative is much more. Of course True positives decrease leads to the decrease in accuracy

Main Disadvantage: Possibility of discarding informative data

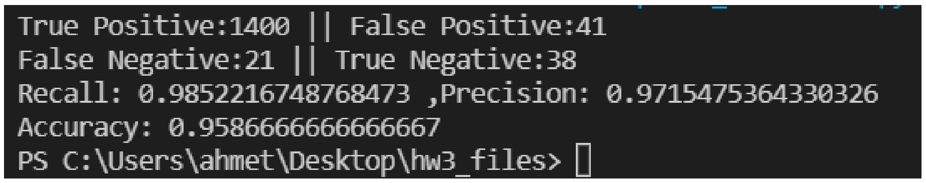
I also wondered the difference undersampling example, the ratio for my second example of undersampling is 242:121 (ratio 2)
This make majoirty class somewhat dominant comparing with ratio 121:121. This leads to increase in accuracy. There is more TP and FP. Due to same reason, there is less FN and TN. This means more majority class(1) prediction is made by model.



```
After undersampling
True Positive:1370 || False Positive:40
False Negative:51 || True Negative:39
Recall: 0.9641097818437719 ,Precision: 0.9716312056737588
Accuracy: 0.9393333333333334
PS C:\Users\ahmet\Desktop\hw3_files> 
```

2.4.4 Setting the class_weight to balanced

Here is my test accuracy, confusion matrix



```
True Positive:1400 || False Positive:41
False Negative:21 || True Negative:38
Recall: 0.9852216748768473 ,Precision: 0.9715475364330326
Accuracy: 0.9586666666666667
PS C:\Users\ahmet\Desktop\hw3_files> 
```

In ths part, I change the class_weigth parameter balance.According to sklearn site, The "balanced" mode uses the values of y to automatically adjust weights inversely proportional to class frequencies in the input data as n_samples / (n_classes * np.bincount(y)).

Therefore, the effect of one data of the majority class decrease while effect of the one data of the minority class increase. This make model balance while learning i.e. model learn under same effect of all class in the data set. It overcome the unbalance class problem. Of course, naturally balanced dataset is better but this is a good solution and there is no disadvantage for computation or memory. I think one disadvantage is decrease for informative data effect

Comparing with experiment1
Recall decrease because there are some false negatives anymore and true positive decrease Precision increase because false positives decrease more comparing with decrease true positives. Accuracy increase because true negative increase more than increase in false negative and decrease in the true positive Also, decrease in false positives promote the accuracy