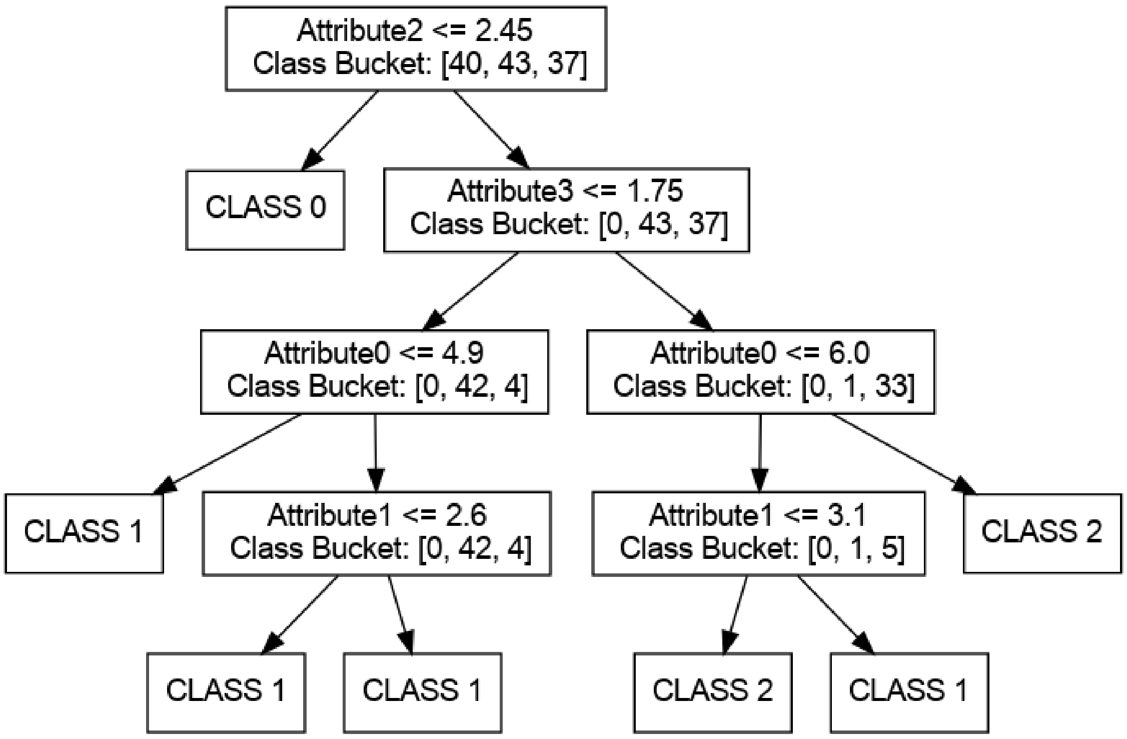
HOMEWORK – 3

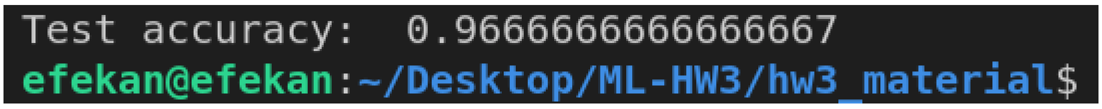
Efekan Çakmak, 2309797

Decision Tree

First, I implemented ID3, and got the decision tree below with training set and 'info_gain' heuristic.



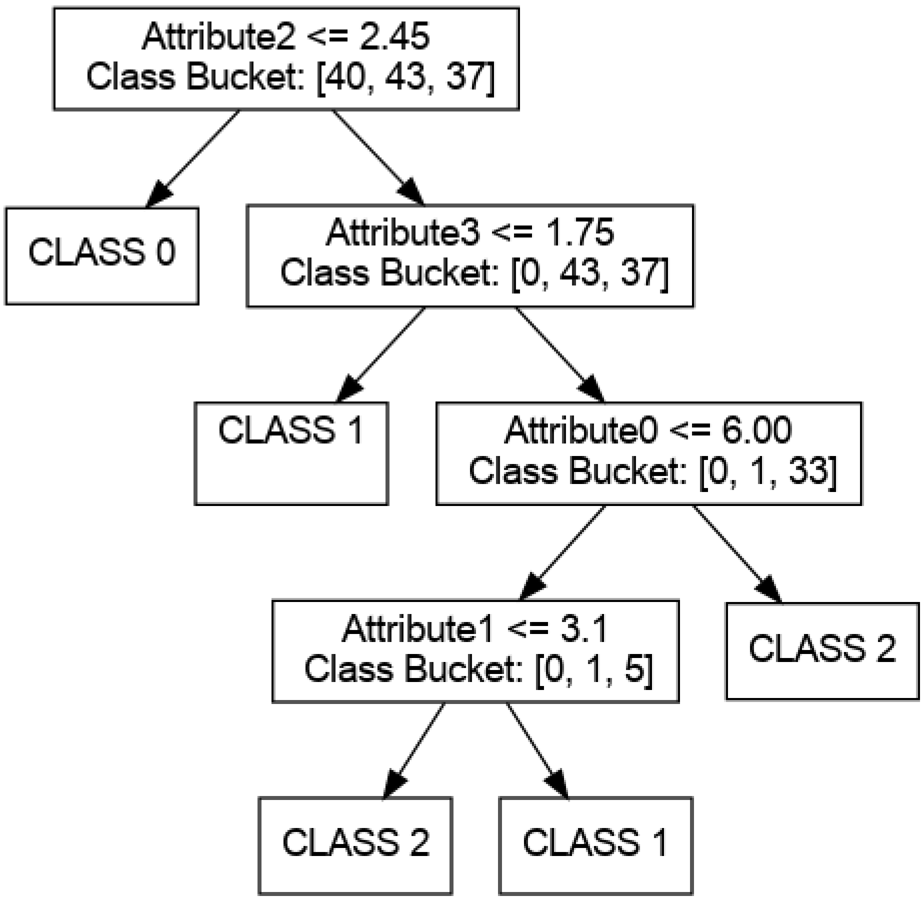
I obtained almost the same tree with 'avg_gini_index' heuristic. One difference is in the deepest and leftmost node. With 'info_gain', split value is 2.6, while it is 2.2 with 'avg_gini_index'.



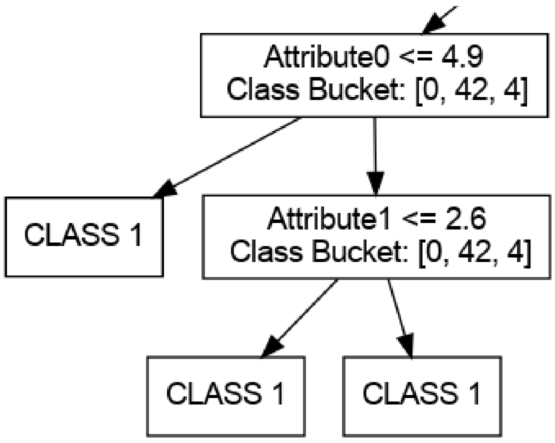
```
Test accuracy:  0.9666666666666667
efekan@efekan:~/Desktop/ML-HW3/hw3_material$
```

Both have 96.66% accuracy on test set.

Then, I created new decision tree by using Chi-Square Test to avoid unnecessary splits.
Both ‘info_gain’ and ‘avg_gini_index’ results in the same tree below.



It is obvious that rightmost nodes are the same with the ones in previous example that we do not use Chi-Square Test. On the other hand, the tree got rid of unnecessary splits.



You can see a part of the tree from previous example. First time, I did not realize that we are splitting two times in vain. Thank to Chi Square test, we got rid of useless branch so that increased our model performance.

Consequently, I got again 96.66% accuracy in both ‘info_gain’ and ‘avg_gini_index’ on test set.

Support Vector Machines

Task 1

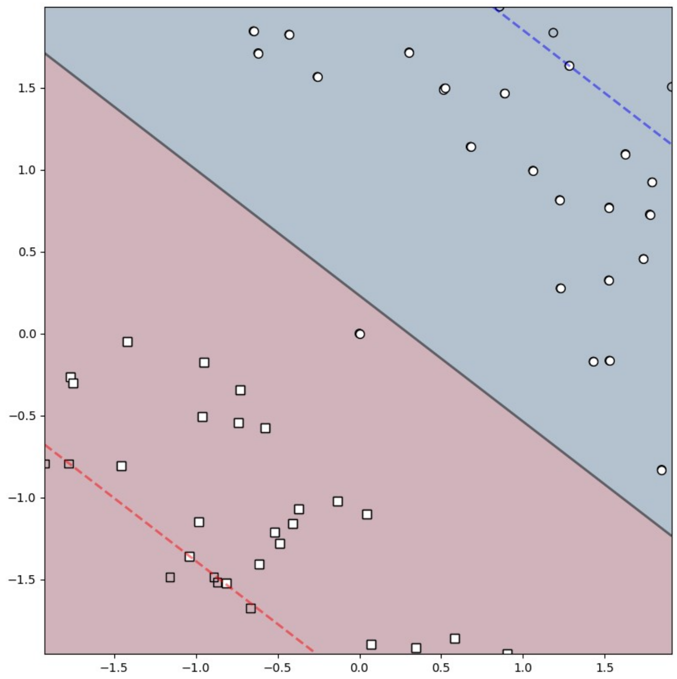


Figure 1: The Linear Classifier with C=0.01

It can be seen from left, our classifier has soft-margin because of small C=0.01 that allows constraints to be easily ignored. Therefore, we have large margin that means that we sacrifice some samples easily. And, we ignore most of them when plotting support lines.

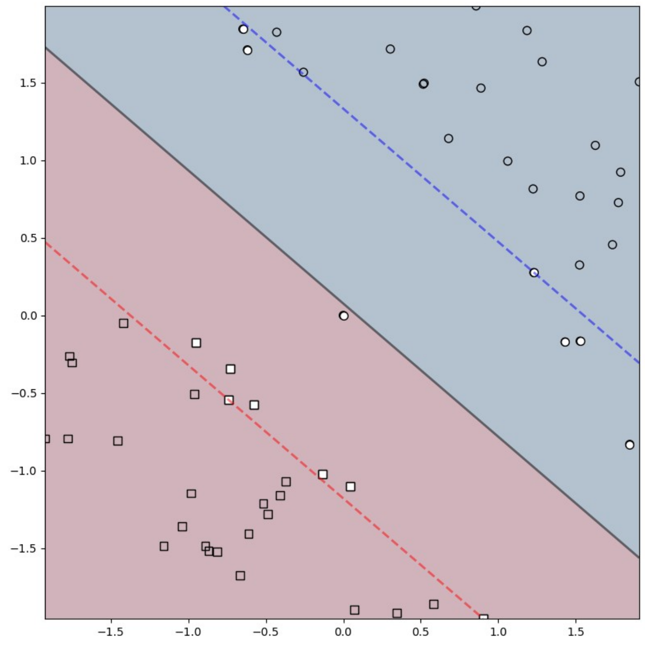


Figure 2: The Linear Classifier with C=0.1

Now we have C=0.1. It can be seen from left, our classifier has a little harder-margin than first classifier, However, it has still soft-margin because of small C=0.1 that allows constraints to be easily ignored. Therefore, we still have large margin that means that we sacrifice some samples, but now we do not ignore most of the examples for plotting support lines.

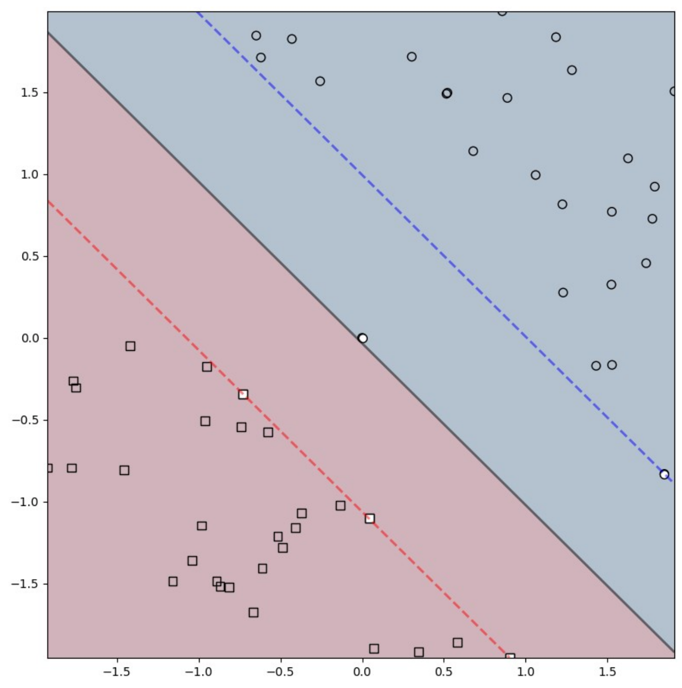


Figure 3: The Linear Classifier with C=1

Now we have C=1. It can be seen from left, our classifier has margin between hard and soft. Therefore, we have narrow margin that means that we do not sacrifice samples, but we ignore some of them when plotting support lines.

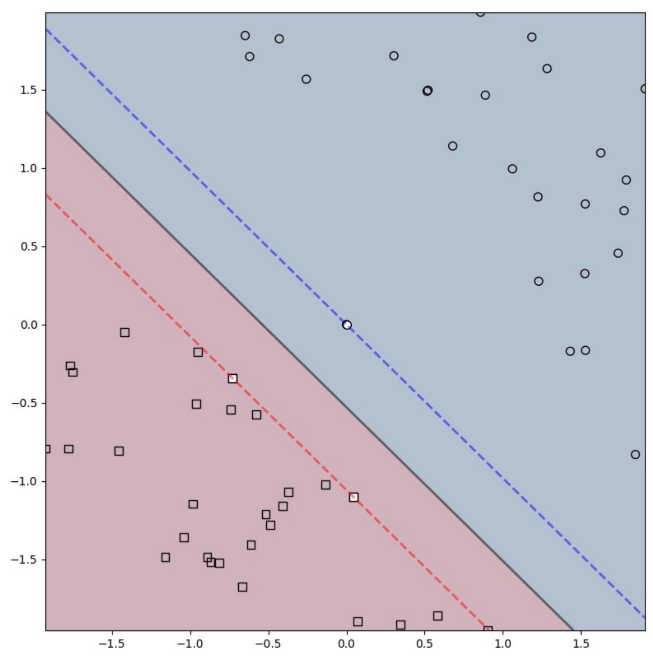
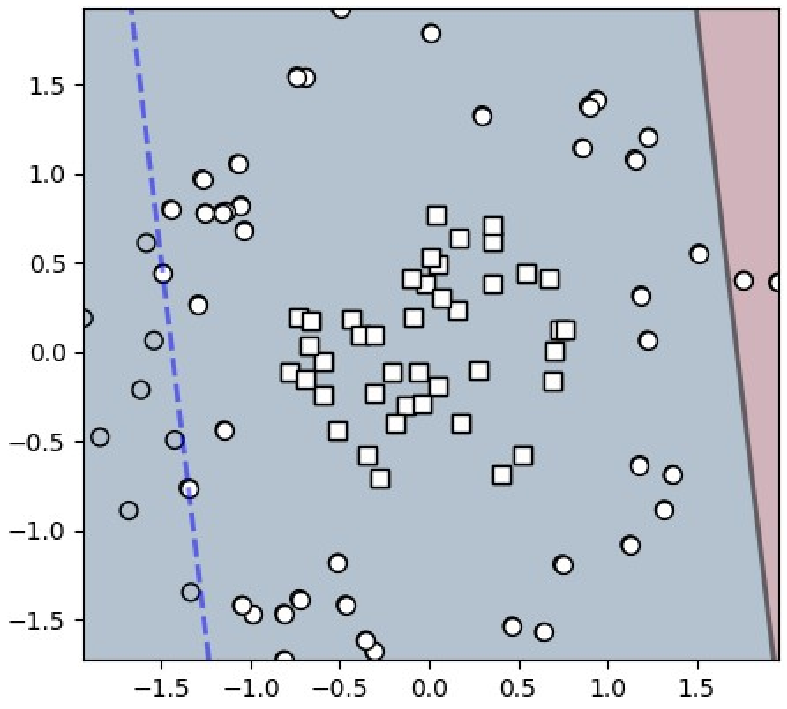


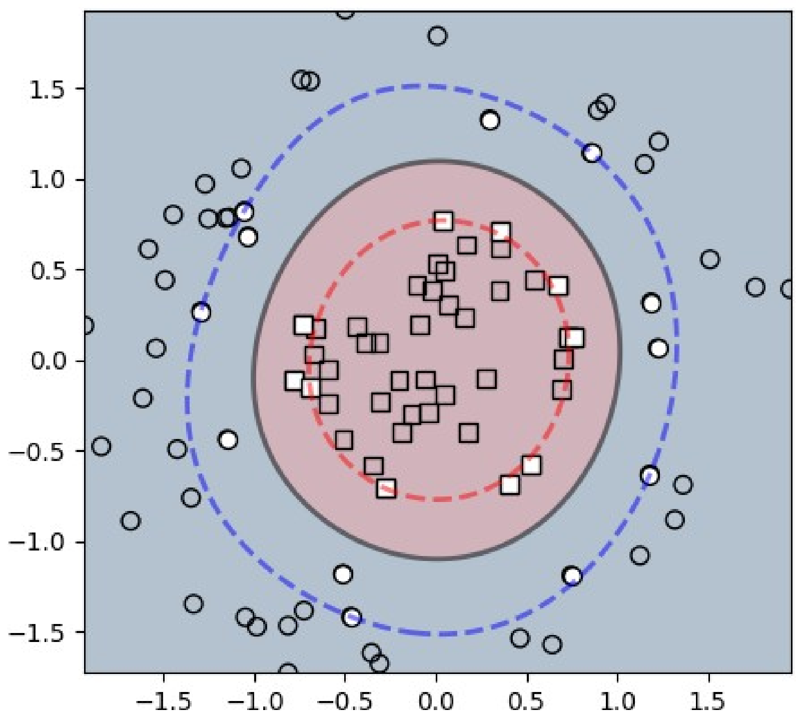
Figure 4: The Linear Classifier with C=10 and C=100

I got the same results from C=10 and C=100. As can be seen left, our classifier has totally hard-margin. Therefore, we have the most narrow margin that means that we certainly do not sacrifice samples, and we do not ignore any of them when plotting support lines.

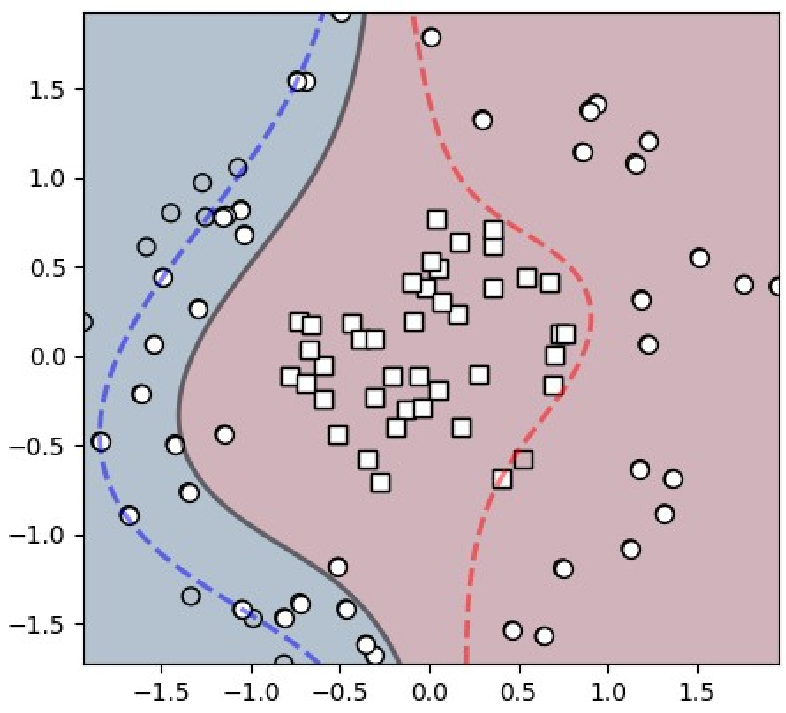
Task 2



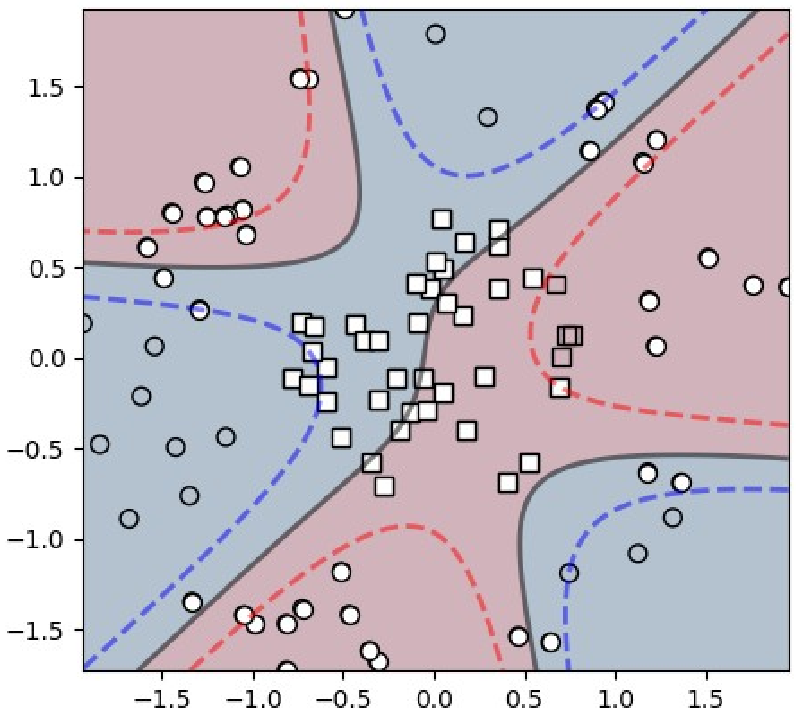
Since we have not linearly-seperable data, using linear kernel to classify this data is irrelevant.
At this point, SVM gives us meaningless output that can be seen left.



I did a short research on RBF and I understood that it is a kernel trick that plots support curves ,instead of support lines, with support vectors. Then I also observed that RBF fits most of various shapes. Consequently, RBF works well in this dataset.



Polynomial Kernel:
K(x,y) = (ax^T y + c)^d
By formula above, it is simply power of line equations. We can achieve curves like y=x^2 , y=x^3 with polynomial kernel. However, we need to closed equation(not one-to-one, not onto) (like circle). Therefore, it does not work well.



It is obvious that sigmoid kernel is pretty much unsuccessful to plot split line. On the other hand, I think support curves are good. Also, I believe if we change its parameters, then we can get better results with sigmoid kernel.

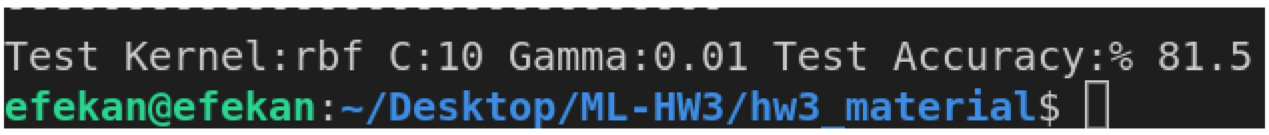
Task 3

The dataset-3 consists of 1000 examples. I used first 800 examples for training and I used last 200 examples to create validation set.

As can be seen below, I used three types of kernels namely linear, rbf with Gamma= 0.01, rbf with Gamma=0.1 and poly. Also, used C parameters 0.001, 0.01, 0.1, 1 and 10. Therefore, I have 20 hyper-parameter configurations.

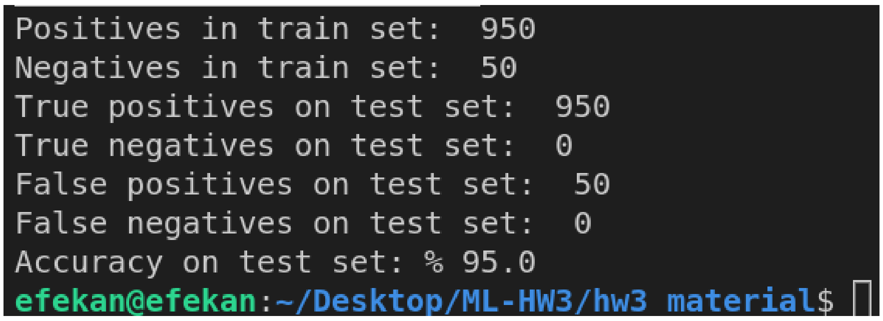
	Kernel	C	Gamma	Validation Accuracy
1	Linear	0.001	-	% 70.0
2	Linear	0.01	-	% 67.0
3	Linear	0.1	-	% 65.0
4	Linear	1	-	% 62.0
5	Linear	10	-	% 61.5
6	Rbf	0.001	0.01	% 42.0
7	Rbf	0.001	0.1	% 42.0
8	Rbf	0.01	0.01	% 42.0
9	Rbf	0.01	0.1	% 42.0
10	Rbf	0.1	0.01	% 74.5
11	Rbf	0.1	0.1	% 42.0
12	Rbf	1	0.01	% 75.5
13	Rbf	1	0.1	% 75.0
14	Rbf	10	0.01	% 78.0
15	Rbf	10	0.1	% 75.0
16	Polynomial	0.001	-	% 72.0
17	Polynomial	0.01	-	% 71.5
18	Polynomial	0.1	-	% 76.0
19	Polynomial	1	-	% 75.0
20	Polynomial	10	-	% 74.5

It is obvious that classifier with kernel Rbf, C=10, and gamma=0.01 yield the best validation accuracy. Its result on test data can be seen below.



```
Test Kernel:rbf C:10 Gamma:0.01 Test Accuracy:% 81.5
efekan@efekan:~/Desktop/ML-HW3/hw3_material$
```

Task 4



```
Positives in train set:  950
Negatives in train set:  50
True positives on test set:  950
True negatives on test set:  0
False positives on test set:  50
False negatives on test set:  0
Accuracy on test set: % 95.0
efekan@efekan:~/Desktop/ML-HW3/hw3 material$
```

Figure 5: Result from SVM with rbf kernel, (C=1)

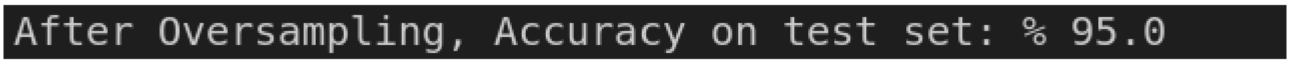
As can be seen above, there is absolute imbalance between classes (1 vs. 19). In this case, if we implement classifier that directly labels new instances as positive or that randomly labels new instances, then the accuracy would converge to 95%.

-	Actual Positive	Actual Negative
Predicted Positive	950	50
Predicted Negative	0	0

Table 1: First Confusion Matrix for Task 4

Obviously, it can be understood from the table above that our classifier, SVM with rbf kernel (C=1), actually labels all new instances as positive. However, since our classes are imbalanced, accuracy misleads us. Therefore, accuracy is NOT good performance metric.

Then, I over-sampled the minority class by selecting randomly from negatives and copying them. There are 950 positives and 950 negatives in the dataset know. Then, I got this results.



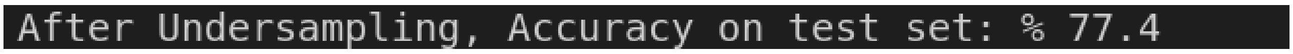
```
After Oversampling, Accuracy on test set: % 95.0
```

-	Actual Positive	Actual Negative
Predicted Positive	938	38
Predicted Negative	12	12

Table 2: Second Confusion Matrix for Task 4

We can see that our classifier has %95 accuracy again, but now it started to detect negatives instead of directly labeling positive. However, it is still bad to detect negatives. Again, accuracy is NOT good performance metric.

Then, I under-sampled the majority class by selecting randomly from positives and deleting them. There are 50 positives and 50 negatives in the dataset know. Then, I got this results.



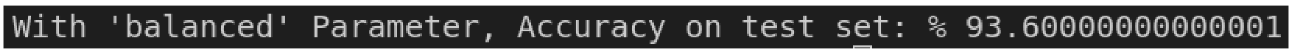
```
After Undersampling, Accuracy on test set: % 77.4
```

-	Actual Positive	Actual Negative
Predicted Positive	737	13
Predicted Negative	213	37

Table 3: Third Confusion Matrix for Task 4

Now, we have %77.4 accuracy. I think that this model under-fits our task because of small size of dataset. On the other hand, I think that accuracy started to say truth to us since this classifier label %78 of the positives and %74 of negatives correctly.

Then, I used parameter configuration in the homework pdf. And, got this result.



```
With 'balanced' Parameter, Accuracy on test set: % 93.60000000000001
```

-	Actual Positive	Actual Negative
Predicted Positive	920	34
Predicted Negative	30	16

Table 4: Fourth Confusion Matrix for Task 4

Now, we have %93.6 accuracy. I think this classifier is better than second classifier to detect negatives, but worse to detect positives. Again, accuracy misleads us, and we should not trust it. We still can not catch almost %70 of negatives.