CENG 499 Special Topics:
Introduction to Machine Learning

HOMEWORK 3 REPORT

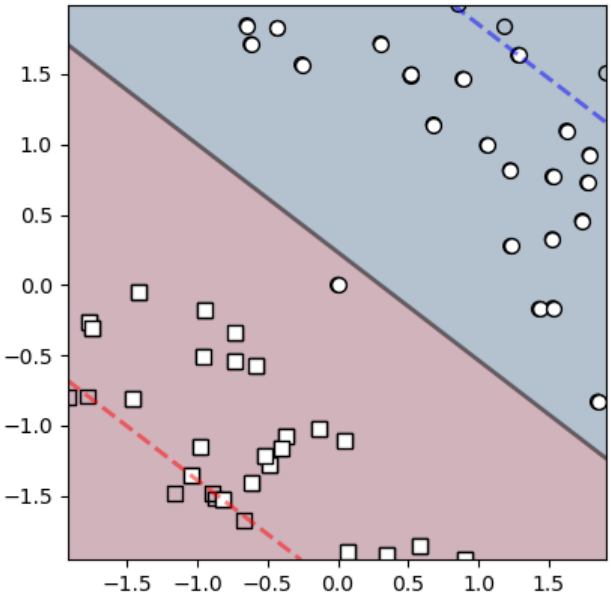
ARDAN YILMAZ

2172195

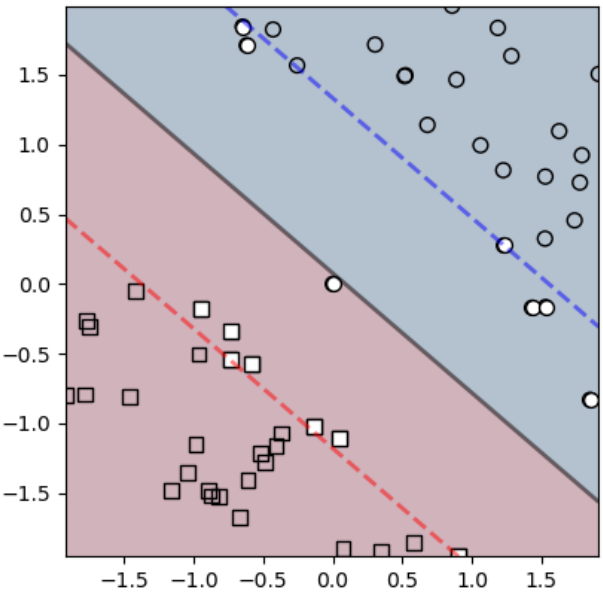
ONLY THE FUNCTIONS IN dt.py HAS BEEN IMPLEMENTED.

SVM TASK 1

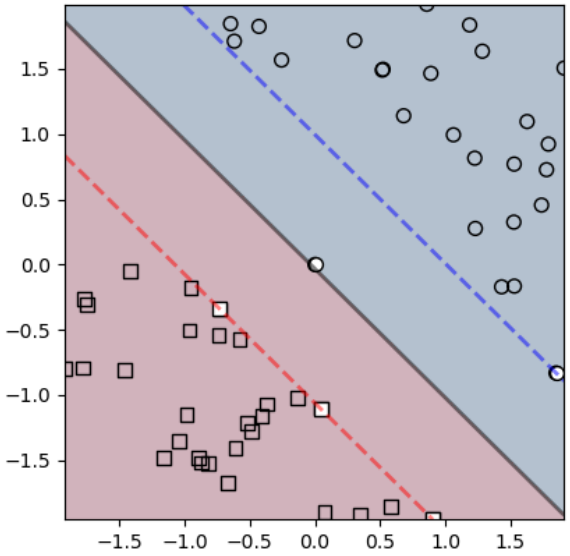
For C = 0.01



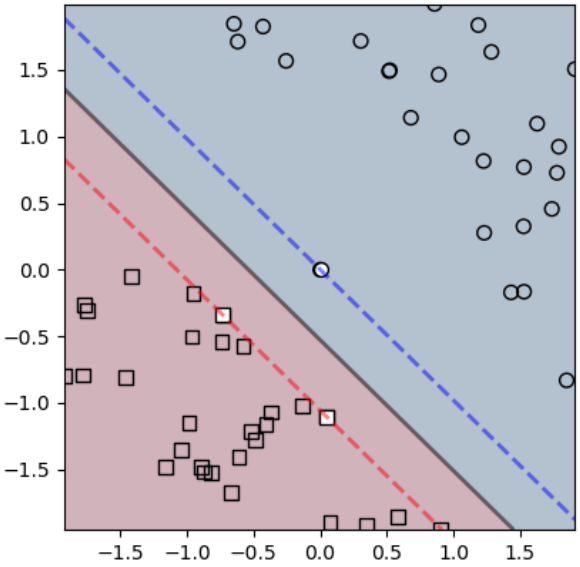
For C = 0.1



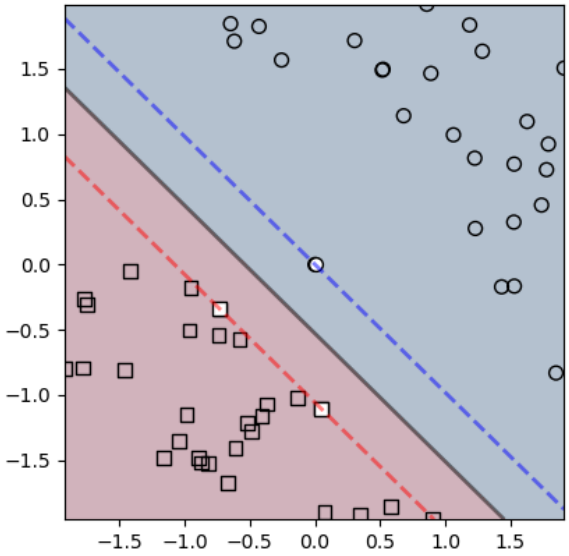
For C = 1



For C = 10



For C = 100

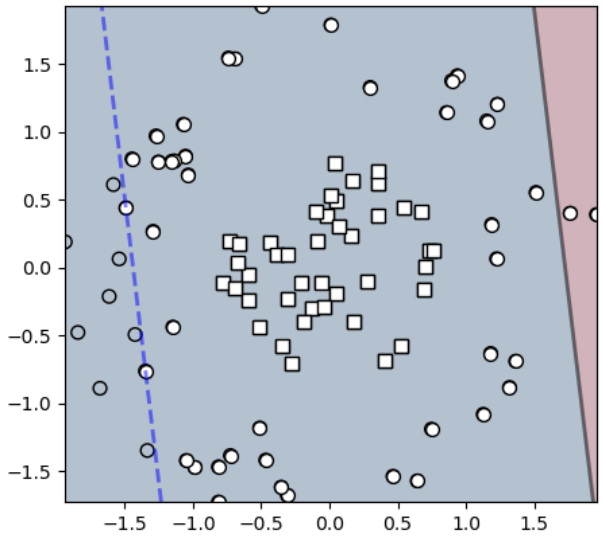


C is the regularization parameter, which is inversely proportional to the strength of regularization. The parameter C is a measure of how much we penalize the SVM classifier for a misclassification. There is naturally a trade-off between how much misclassification is allowed and the margin. In other words, the objective function for SVM both tries to minimize the error, ie, misclassification, and maximize the margin. Penalizing the classifier more on misclassification (with greater C values), there is a stricter limit on the margin maximization. And, one can observe these clearly with the given classifier plots above. That is, with greater C values the margin gets more and more narrow while for a small C, such as 0.01, even misclassification is allowed.

There is a circular data point in the blue region that is rather far away from its own class, it might be a better idea to sacrifice it, ie, misclassify it for the sake of generalization, ie, avoiding overfitting. Hence, C = 1 appears to be a reasonable choice.

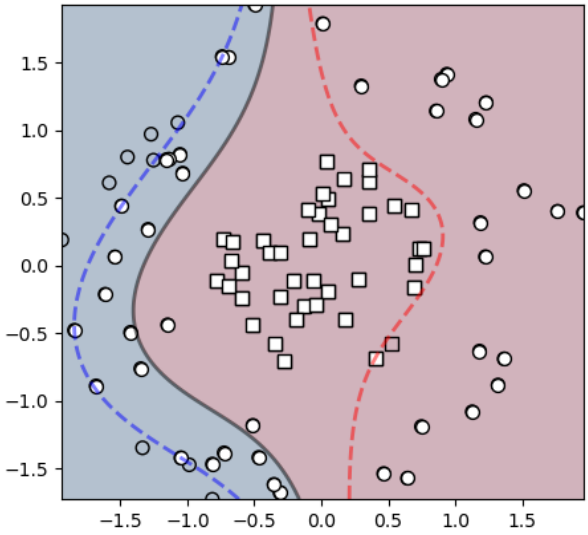
SVM TASK 2

For linear kernel:



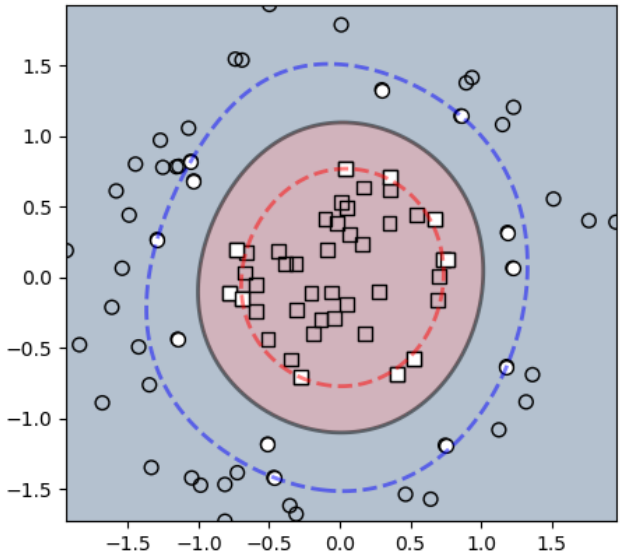
The dataset is linearly inseparable, so the linear kernel failed to classify it.

For polynomial kernel:



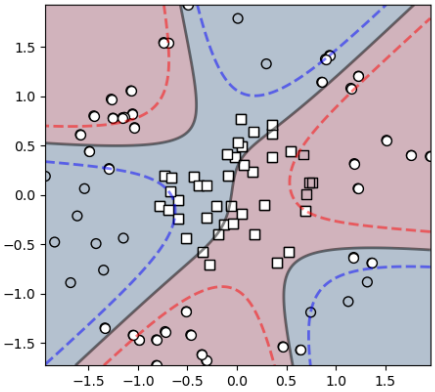
The data points cannot be classified with a polynomial classifier either, as they are clustered circularly around each other. Hence, a circular classifier is needed to separate these.

For RBF kernel:



RBF kernel did the trick to classify such distribution of data points. RBF gives access to non-parametric functions as well, which is why it did well to classify this data while the polynomial kernel failed.

For sigmoid kernel:



Sigmoid kernel failed to classify the data as can be seen.

SVM TASK 3

Hyperparameter tuning results:

Kernel	C	gamma	Validation Accuracy
linear	0.01	0.001	**0.74**
linear	0.01	0.01	**0.74**
linear	0.01	0.1	**0.74**
linear	0.01	1	**0.74**
linear	0.1	0.001	**0.73**
linear	0.1	0.01	**0.73**
linear	0.1	0.1	**0.73**
linear	0.1	1	**0.73**
linear	1	0.001	**0.745**
linear	1	0.01	**0.745**
linear	1	0.1	**0.745**
linear	1	1	**0.745**
linear	10	0.001	**0.74**
linear	10	0.01	**0.74**
linear	10	0.1	**0.74**
linear	10	1	**0.74**
linear	100	0.001	**0.74**
linear	100	0.01	**0.74**
linear	100	0.1	**0.74**
linear	100	1	**0.74**

Kernel	C	gamma	Validation Accuracy
polynomial	0.01	0.001	**0.49**
polynomial	0.01	0.01	**0.725**
polynomial	0.01	0.1	**0.83**
polynomial	0.01	1	**0.81**
polynomial	0.1	0.001	**0.565**
polynomial	0.1	0.01	**0.79**
polynomial	0.1	0.1	**0.81**
polynomial	0.1	1	**0.81**
polynomial	1	0.001	**0.7**
polynomial	1	0.01	**0.775**
polynomial	1	0.1	**0.81**
polynomial	1	1	**0.81**
polynomial	10	0.001	**0.725**
polynomial	10	0.01	**0.83**
polynomial	10	0.1	**0.81**
polynomial	10	1	**0.81**
polynomial	100	0.001	**0.79**
polynomial	100	0.01	**0.81**
polynomial	100	0.1	**0.81**
polynomial	100	1	**0.81**

Kernel	C	gamma	Validation Accuracy
RBF	0.01	0.001	**0.49**
RBF	0.01	0.01	**0.49**
RBF	0.01	0.1	**0.49**
RBF	0.01	1	**0.49**
RBF	0.1	0.001	**0.69**
RBF	0.1	0.01	**0.775**
RBF	0.1	0.1	**0.5**
RBF	0.1	1	**0.49**
RBF	1	0.001	**0.775**
RBF	1	0.01	**0.83**
RBF	1	0.1	**0.75**
RBF	1	1	**0.49**
RBF	10	0.001	**0.785**
RBF	10	0.01	**0.845**
RBF	10	0.1	**0.755**
RBF	10	1	**0.49**
RBF	100	0.001	**0.825**
RBF	100	0.01	**0.865**
RBF	100	0.1	**0.49**
RBF	100	1	**0.49**

Kernel	C	gamma	Validation Accuracy
sigmoid	0.01	0.001	**0.49**
sigmoid	0.01	0.01	**0.49**
sigmoid	0.01	0.1	**0.49**
sigmoid	0.01	1	**0.49**
sigmoid	0.1	0.001	**0.685**
sigmoid	0.1	0.01	**0.405**
sigmoid	0.1	0.1	**0.49**
sigmoid	0.1	1	**0.49**
sigmoid	1	0.001	**0.755**
sigmoid	1	0.01	**0.675**
sigmoid	1	0.1	**0.49**
sigmoid	1	1	**0.49**
sigmoid	10	0.001	**0.72**
sigmoid	10	0.01	**0.675**
sigmoid	10	0.1	**0.495**
sigmoid	10	1	**0.49**
sigmoid	100	0.001	**0.655**
sigmoid	100	0.01	**0.68**
sigmoid	100	0.1	**0.49**
sigmoid	100	1	**0.49**

The hyperparameter configuration to yield the highest accuracy, ie, kernel = rbf, C=100, gamma = 0.01, is highlighted with red on the table.

When the classifier is trained with the whole dataset and the above hyperparameters, the resulting test accuracy is 0.508.

SVM TASK 4

The following confusion matrices are of the following form:
[[TN FP
FN TP]]

4.1.

Accuracy: 0.947

Confusion matrix:

[[2 48]
[5 945]]

Accuracy is not a feasible metric for such a class imbalance case, as labeling everything positive yields extremely high accuracy results. Accuracy does not take false positives into count. Additionally, the test set has the same imbalance, that's why the commonly used metrics seem to yield good results. However, checking the Matthews Correlation Coefficient (MCC) value, for example, is around 0.09, which is almost the predictive performance of its random classifier. That is, training a model with mostly positive instances and testing with a similar case obviously yields almost perfect accuracy, which gives almost no information about how well the model does on negative predictions.

4.2.

Oversampling the minority class, ie, labeled 0

Accuracy: 0.936
Confusion matrix:

[[21 29]
[35 915]]

Although oversampling the minority class yielded slightly poorer accuracy than the first case, we can see the drop in the false positives. The test set is still imbalanced, so it is hard to say something about the bias, however, compared with the previous one it is almost certain that this classifier is better.
Again, for this kind of class imbalance, accuracy is not a good metric. The resultant MCC score for this one is 0.36, which is clearly better than that of the first.

4.3.

Undersampling the majority class, ie, labeled 1

Accuracy: 0.779

Confusion matrix:

[[24 26]
[195 755]]

The accuracy is lower than those of the previous methods due to the drop in the number of true positives. Still, compared with the first method, this one is more reliable on negative predictions.

4.4.

Setting the class_weight parameter = “balanced”.

Accuracy: 0.928

Confusion matrix:

[[22 28]
[44 906]]

There is a significant drop in the number of false positives with this method as in the second case. Again, although with a poorer accuracy, this can be safely said to be a better classifier than the first method, especially on the negative predictions.

Briefly, accuracy is not a feasible metric to evaluate the predictive performance of such models trained and tested with imbalanced classes. Even if the model is trained with a balanced data set, we calculate the performance results on test sets, and the test sets are all imbalanced here, hence metrics that can easily be tricked such as accuracy is potentially misleading.