Tuan Tran
A20357888
CS584
Summer 2020

A. Implementation details:

The program is divided into 3 total relevant files. There are 2 notebooks, the first notebook contains code that returns results related to the synthetic dataset while the second notebook contains results for the external dataset. Each cell in each notebook also has been fully documented and should run fine on any machine with the appropriate python modules. To use the program and check for correctness of the results stated, simply go into the notebook and run each of the cells.

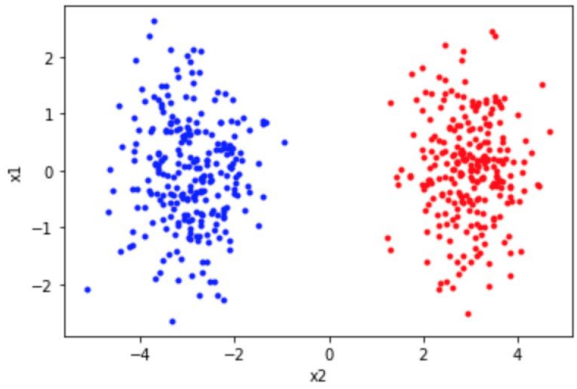
Regarding data, for the generated synthetic data, there are 4 files in total which represents the combination of class 0/1 and linearly separable/linearly non-separable. Please simply run the notebook cell to get the result, there's no need to do anything for these files. For the external data, please download the data following the links from the data folder, and place the data files in the same directory as the notebooks.

The implementation.py file contains the models implementations and helper functions. Inside are the functions for plotting the decision function, margin and support vectors, as well as three different models: Hard Margin SVM, Soft Margin SVM, and Soft Margin Kernel SVM with options of polynomial or gaussian kernel. Each of the models has 3 main functions which are fit(), predict() and decision_function() to help with plotting.

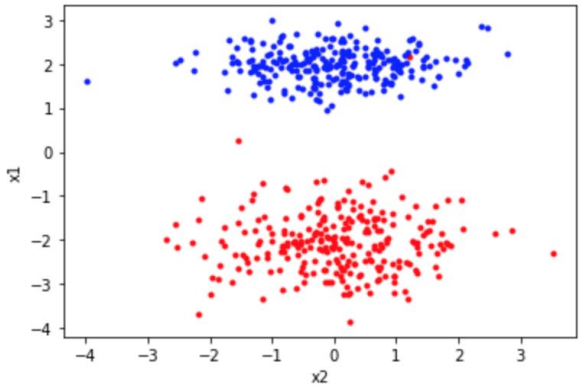
B. Result and Discussion:

1. Synthetic Data Generation:

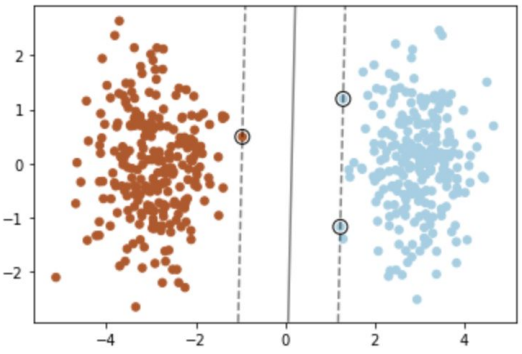
We first generated a dataset of 500 samples in total which are very linearly separable by creating datasets by make_classification method from scikit-learn and stop when we find a dataset that is linearly separable (we tested for vertical and horizontal separation, which is enough since it's a 2D feature vectors dataset). We intentionally made this dataset very separable so that it serves as a very basic correctness check of each SVM implementation that we tested. We saved this dataset under the name of "linearly_separable_classx.npy" where x = 0 or 1. The dataset is as follows:



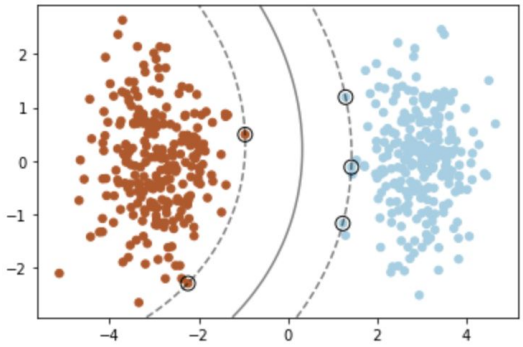
We then generated a dataset of 500 samples that are linearly non-separable using the same method. This data has one red point in a cluster of blue so it's a good starting point:



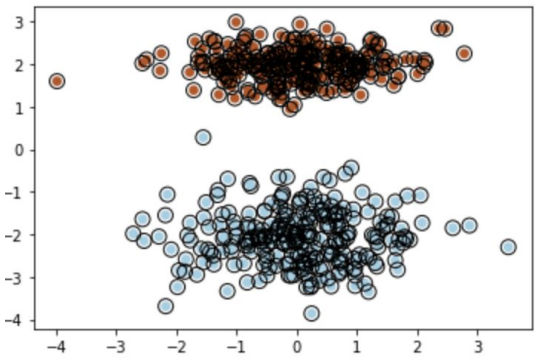
2. Synthetic Data Experiment:
   On the linearly separable data, all SVM models did a good job at separating the classes and identifying the support vectors. For example, the hard margin SVM yielded the following result:



As another example of a different shape decision boundary, the polynomial SVM with degree 2 yielded the following curve:

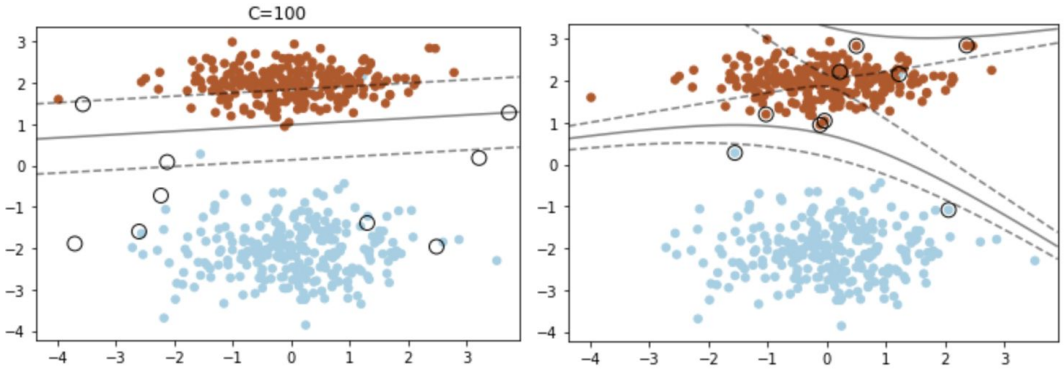


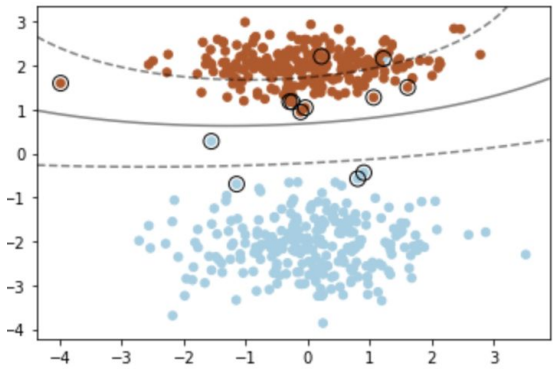
However, in the case of linearly inseparable data, it is not surprising that it gave a much harder time for all of the models. In fact, hard margin SVM couldn't find any solution to the optimization problem due to its nature and it terminated upon singular KKT matrix, yielding:



We can see that the alphas solution returned is trivial, the solver dictates that all points are support vectors.

Both soft margin SVM and kernel SVM (implemented as soft margin) were able to find a solution. The decision boundaries were good separation between the classes, but unfortunately didn't obtain the outside-of-margin solution. The following graphs are results from soft margin SVM, polynomial SVM (degree = 2) and gaussian SVM respectively:





For the soft margin SVM, we also tried varying the C hyperparameter between the values of 10, 100, 500 and 1000. However, surprisingly, 10 and 100 didn't yield too different results, while 500 and 1000 yielded results that were too noisy, thus we decided not to include the graphs for other values of C in this report. It's not surprising that all these models cannot find a good solution to this dataset. We can see that the blue point is too clustered in with the brown point cluster, thus an exact separation with good margin is almost impossible.

3. External Data Experiments:
   For external data, we chose the Abalone dataset where the goal is to predict the number of rings of an abalone given measurements and sex (nominal feature). We turn this into a 2-class classification problem by first choosing the subset of 2 classes with the most number of instances. We then encode these 2 labels into -1 and 1 in order to feed into our implementation of SVM. Finally we one-hot encode the nominal feature and transform the data into training matrices.

   First, we use 5-fold cross validation to compare the polynomial model vs the gaussian model in their respective default setting (degree = 2 for polynomial and sigma = 5.0 for gaussian). We observed the following results averaging the accuracy score across the folds (we use accuracy score since the data is relatively balanced):

```
SVM with polynomial kernel:
        Average training accuracy score: 63.0197550565769
        Average testing accuracy score: 61.90308747855917
SVM with radial kernel:
        Average training accuracy score: 59.35386845313641
        Average testing accuracy score: 58.3516295025729
```

   We can see that the polynomial SVM is slightly better than the gaussian SVM on average, however both models did pretty badly on the classification task. We now rely on changing the hyperparameters of the models, increasing its complexity to see if we can improve the accuracy.

For the polynomial SVM model, we vary the degree from degree = 2 to 4 (1 increment), making the model more complex by the increase of degree. Again using 5-fold cross validation and averaging accuracy across the folds, we obtain:

```
Polynomial SVM with degree 2:
        Average training accuracy score: 63.0197550565769
        Average testing accuracy score: 61.90308747855917
Polynomial SVM with degree 3:
        Average training accuracy score: 64.1533100920903
        Average testing accuracy score: 60.09005145797599
Polynomial SVM with degree 4:
        Average training accuracy score: 64.87159302477102
        Average testing accuracy score: 59.63893653516295
```

Quite surprisingly, increasing the degree only slightly does not improve the performance at all. In fact, we can see that the training accuracy increases slightly, but the testing accuracy actually decreases. This signifies that the model is overfitting, instead of improving overall performance. Indeed, increasing the degree to 6, we observe decent increase in training accuracy but also significant decrease on testing accuracy:

```
Polynomial SVM with degree 6:
        Average training accuracy score: 67.93308235646926
        Average testing accuracy score: 56.31389365351629
```

Similarly, for the gaussian model, we vary sigma from 1.0 to 15.0 (5.0 increment) and observed the following result:

```
Gaussian SVM with sigma 1.0:
        Average training accuracy score: 61.961457379451666
        Average testing accuracy score: 60.4685534591195
Gaussian SVM with sigma 5.0:
        Average training accuracy score: 59.35386845313641
        Average testing accuracy score: 58.3516295025729
Gaussian SVM with sigma 10.0:
        Average training accuracy score: 57.9555381811496
        Average testing accuracy score: 57.066895368782156
Gaussian SVM with sigma 15.0:
        Average training accuracy score: 57.35090885398537
        Average testing accuracy score: 56.688965122927385
```

We observed that increasing sigma also hurts performance. We hypothesize that the bad performance is due to the nature of the data itself, how we frame the problem and most importantly for our task at hand, the subset of class chosen. The original dataset label has a lot of values, since the target is the number of rings of each abalone, thus it is better suited as a regression problem, than a classification problem. However, we still try to frame it as a classification problem since the values are discrete, and further only choose a subset of 2 classes to try to predict. So the problem may be ill-formed. Moreover, and **more importantly**, the subset chosen were **class 9 and 10** (so number

of rings = 9 and = 10) since they have the most number of instances. We can immediately see how framing the problem as classification with this particular class subset (9 and 10) is problematic, since practically and realistically, classifying 9 vs 10 rings is **very hard** as the measurements/features may be almost similar (since an abalone with 9 rings is very similar to the one with 10 rings).

We decided to repeat **all performance tests** with a different subset of 2 classes which are further apart: class 7 (7 rings) and 11 (11 rings). Indeed, we get a **huge performance improvement** across **all experiments**:

Polynomial SVM vs Gaussian SVM:

```
SVM with polynomial kernel:
        Average training accuracy score: 91.51491572544204
        Average testing accuracy score: 89.52597402597404
SVM with radial kernel:
        Average training accuracy score: 86.44664907822802
        Average testing accuracy score: 85.19350649350649
```

Polynomial SVM with varying degree:

```
Polynomial SVM with degree 2:
        Average training accuracy score: 91.51491572544204
        Average testing accuracy score: 89.52597402597404
Polynomial SVM with degree 3:
        Average training accuracy score: 92.08475682159893
        Average testing accuracy score: 90.09675324675325
Polynomial SVM with degree 4:
        Average training accuracy score: 92.4264345316977
        Average testing accuracy score: 89.18636363636365
```

Gaussian SVM with varying sigma:

```
Gaussian SVM with sigma 0.1:
        Average training accuracy score: 95.55863555863556
        Average testing accuracy score: 88.61363636363636
Gaussian SVM with sigma 1.0:
        Average training accuracy score: 90.14840751682857
        Average testing accuracy score: 88.49805194805197
Gaussian SVM with sigma 5.0:
        Average training accuracy score: 86.44664907822802
        Average testing accuracy score: 85.19350649350649
Gaussian SVM with sigma 10.0:
        Average training accuracy score: 85.25067577699156
        Average testing accuracy score: 84.28051948051947
```