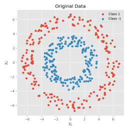
Amin Fadaee

Machine Learning HW4

Python 3.5, Libraries: LIBSVM, Numpy, Matplotlib, Pandas, Math, random, sklearn Documentation: HTML, Bootstrap 4, jqMath, Graphs: XMind

Problem 1

a. Here is a plot depicting the data:

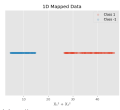


b. To transform the data into a 1-D linearly separable space, we can use the following transformation function:

$$f_1(x_1, x_2) = x_1^2 + x_2^2$$

The reason for choosing this transformation is the circular shape of the data in each class and the fact that the data in different classes have a distinguishable distance from the origin. Although for simplicity we do not perform the square operation but this will not cause any harm.

Here is the transformed data:

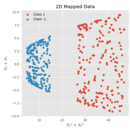


As can be seen, the data is now perfectly separable.

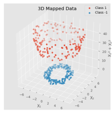
c. While preserving the feature we created in the previous part, we can add another feature that simply sums the two features:

$$f_2(x_1, x_2) = (x_1 + x_2, x_1^2 + x_2^2)$$

Doing so would result in the following transformed data:



d. As for the choice of transforming to 3D, we can just add another feature ($f_3(x_1, x_2) = (x_1, x_2, x_1^2 + x_2^2)$) to the original data and deriving the following new data:



e. We can derive the kernels of the mentioned transformation functions in the following way ($x = x_1, y = x_2, X = [x, y]^T$ and $K(X_i, X_j) = X_i^T X_j$):

1. $f_1(x, y) = x^2 + y^2$
   $$K_1(X_i, X_j) = (x_i^2 + y_i^2)(x_j^2 + y_j^2)$$
2. $f_2(x_1, x_2) = (x_1 + x_2, x_1^2 + x_2^2)$
   $$K_2(X_i, X_j) = (x_i + y_i)(x_j + y_j) + (x_i^2 + y_i^2)(x_j^2 + y_j^2)$$
3. $f_3(x_1, x_2) = (x_1, x_2, x_1^2 + x_2^2)$
   $$K_3(X_i, X_j) = x_i x_j + y_i y_j + (x_i^2 + y_i^2)(x_j^2 + y_j^2)$$

Now we can use these kernels for further analysis. In `Problem 1/Part 5.py` we used them to derive the Kernel Matrix of data for each of the transformations (these kernel matrices are saved into .csv files `prek1.csv, prek2.csv, prek3.csv` for later use).

f. After training a Linear SVM on the original data using 5-fold cross validation the following final result were obtained:

```
optimization finished, #iter = 2694
nu = 0.954630
obj = -305.456840, rho = 0.327684
nSV = 307, nBSV = 304
Total nSV = 307
Cross Validation Accuracy = 50.5%
```

As can be seen the model achieved 50.5% accuracy which were expected considering the shape of the data and trying to separate it using a simple line.

g. Now let's use the same model but this time on the data we produced in part b,c and d. After using the same options for LIBSVM (5-fold CV, linear kernel) here is the final result for each of the data:

1-D Data (b):

```
optimization finished, #iter = 3
nu = 0.000089
obj = -0.014244, rho = 3.479454
nSV = 2, nBSV = 0
Total nSV = 2
Cross Validation Accuracy = 100%
```

2-D Data (c):

```
optimization finished, #iter = 50
nu = 0.000089
obj = -0.014168, rho = 3.471120
nSV = 3, nBSV = 0
Total nSV = 3
Cross Validation Accuracy = 100%
```

3-D Data (d):

```
optimization finished, #iter = 74
nu = 0.000086
obj = -0.013750, rho = 3.434650
nSV = 4, nBSV = 0
Total nSV = 4
Cross Validation Accuracy = 100%
```

As can be seen the result for all 3 data is perfect (100% Accuracy). This is due to the fact that using the provided mapping function we were able to make the data linearly separable and therefor a linear separator can perform well with them.

h. Using the kernel matrix we saved in e, we can use LIBSVM feature namely precomputed kernels and use these matrices as the training data. Doing so with (again) 5-fold CV will result in:

Kernel Matrix 1:

```
optimization finished, #iter = 3
nu = 0.000089
obj = -0.014244, rho = 3.479454
nSV = 2, nBSV = 0
Total nSV = 2
Cross Validation Accuracy = 100%
```

Kernel Matrix 2:

```
optimization finished, #iter = 50
nu = 0.000089
obj = -0.014168, rho = 3.471120
nSV = 3, nBSV = 0
Total nSV = 3
Cross Validation Accuracy = 100%
```

Kernel Matrix 3:

```
optimization finished, #iter = 74
nu = 0.000086
obj = -0.013750, rho = 3.434650
nSV = 4, nBSV = 0
Total nSV = 4
Cross Validation Accuracy = 100%
```

As expected, the result are completely corespondent with that of the previous section.

Problem 2

a.
1. We split the Prakinsons data set into training (60%), validation (20%) and test set (20%). Using RBF kernal and the validation set, different values for C and Gamma(10^{-5} to 10^5) has been tested. Here is the final result:

   ```
   Best Validation Result: C=100.0, gamma=1e-05, Accuracy=92.30
   ```

   Using $Gamma = 1e-5$ and $C = 100$ as the final hyperparamters, here is the accuracy obtained on classifying the test set:

   ```
   Test Accuracy: 84.61
   ```
2. Alse 20 randomly generated values for the parameters (from the range 10^{-5} to 10^5) have been used in the same way which resulted in:

   ```
   Best Validation Result: C=100000000.0, gamma=0.0001, Accuracy=87.17
   ```

   And plugging $C = 100000000$ and $Gamma = 0.0001$, the test accuracy is obtained:

   ```
   Test Accuracy: 69.23
   ```
3. Using grid search to find the parameters is exhaustive and time consuming, on the other hand random search might achieve a good result and may as well fail. Using an optimization procedure like bayesian optimization or gradient descent might help finding good parameters faster.

b. For the polynomial kernel, after scaling the data we used the following parameters $d \in 1, 2, 3, 4, 5, 6, 7, 8, 9, 10$ and Gamma in range 10^{-5} to 10^5 in a grid search to find the best ones via validation analysis. Here is the result:

```
Best Validation Result: degree= 4, gamma=1.0, Accuracy=100.0
```

As can be seen the accuracy is 100% for $D = 4$ and $Gamma = 1$, using these parameters we can obtain the test error:

```
Test Accuracy: 82.05
```

Now using 20 random generated set of parameters for $C, gamma, coef0$ in range 10^{-10} to 10^{10} and $d \in 1, 2, 3, 4, 5, 6, 7, 8, 9, 10$, the validation accuracy:

```
Best Validation Result: C=100000.0, gamma=1.0, degree=4, coef0=1.0, Accuracy=100.0
```

And using the obtained parameters we get the test accuracy:

```
Test Accuracy: 87.17
```

c. For the sigmoid kernel, after again scaling the data we used the following parameters C, coef0 and Gamma in range 10^{-5} to 10^5 in a grid search to find the best ones via validation analysis. Here is the result:

```
Best Validation Result: C=10.0, gamma=0.01, coef0=0.1, Accuracy=94.87
```

Using these best parameters we can obtain the test error:

```
Test Accuracy: 87.17
```

Now using 20 random generated set of parameters for $C, gamma, coef0$ in range 10^{-10} the validation accuracy:

```
Best Validation Result: C=0.01, gamma=1000000000.0, coef0=100000.0, Accuracy=79.48
```

And using the obtained parameters we get the test accuracy:

```
Test Accuracy: 76.92
```

d. The C parameter tells the SVM optimization how much we want to avoid misclassifying each training example. For large values of C, the optimization will choose a smaller-margin hyperplane if that hyperplane does a better job of getting all the training points classified correctly. Conversely, a very small value of C will cause the optimizer to look for a larger-margin separating hyperplane, even if that hyperplane misclassifies more points. For very tiny values of C, we get misclassified examples, often even if the training data is linearly separable.

We put this to the test by training a polynimial data with gamma=1, D=4 and C=10e-20 and 10e+20.

C=10e-20

```
Test Accuracy: 76.92
```

C=10e+20

```
Test Accuracy: 84.61
```

e. And finally here is the number of support vectors for each of the procedures we did in section a through c:

a.
1. Number of Support Vectors: 53
2. Number of Support Vectors: 40

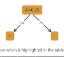
b.
1. Number of Support Vectors: 65
2. Number of Support Vectors: 53

c.
1. Number of Support Vectors: 51
2. Number of Support Vectors: 62

Problem 3

a. Adaboost is sensitive to noise. This is due to the fact that if something is deeply in the positive class region but actually labeled as being in the negative class it would suffer a large loss/penalty since the penalty is exponentiated, so this one outlier/mislabeled point could end up having a very strong influence on the final model learned.

b. Given the data, in the first step we choose the best base classifier which is:



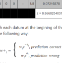
Using this classifier would result in 2 misclassification which is highlighted in the table below:

ID	A	B	Class	w	w+
X1	-1	0	+	1/8	0.07216878
X2	-0.5	0.5	+	1/8	0.07216878
X3	0	1	-	1/8	0.07216878
X4	0.5	1	-	1/8	0.07216878
X5	1	0	+	1/8	0.21650635
X6	1	-1	+	1/8	0.21650635
X7	0	-1	-	1/8	0.07216878
X8	0	0	-	1/8	0.07216878
					z = 0.866025403784

In the table above, w is the weights associated with each datum at the begining of the iteration and $w+$ is the unnormalized weight after we apply the classifier. $w+$ is computed in the following way:

$$w+_i = \begin{cases} w_i e^{-\alpha_t}, \textit{prediction correct} \\ w_i e^{\alpha_t}, \textit{prediction wrong} \end{cases}$$

Where $\alpha = \frac{1}{2} ln\left(\frac{1-e}{e}\right)$ which in this iteration would be $\alpha_1 = 0.54930614433405478$ because $e_1 = 0.25$. Using $z_1 = 0.866025403784$ as a normalizing factor for the weights and using the next base classifier,

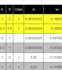


we would obtain the following:

ID	A	B	Class	w	w+
X1	-1	0	+	0.08333333	0.186339
X2	-0.5	0.5	+	0.08333334	0.186339
X3	0	1	-	0.08333335	0.0372678
X4	0.5	1	-	0.08333336	0.0372678
X5	1	0	+	0.25	0.1118034
X6	1	-1	+	0.25	0.1118034
X7	0	-1	-	0.08333339	0.0372678
X8	0	0	-	0.08333340	0.0372678
					z = 0.7453559925

With $\alpha_2 = 0.80471895621705014$, $e_2 = 0.166666666667$ and $z_2 = 0.7453559925$

In the final iteration, using this classifier:



The following table is obtained:

ID	A	B	Class	w	w+
X1	-1	0	+	0.25	0.08333333
X2	-0.5	0.5	+	0.25	0.08333333
X3	0	1	-	0.05	0.01666667
X4	0.5	1	-	0.05	0.01666667
X5	1	0	+	0.15	0.05
X6	1	-1	+	0.15	0.05
X7	0	-1	-	0.05	0.15
X8	0	0	-	0.05	0.15
					z =0.6

With $\alpha_3 = 1.0986122886681$, $z_3 = 0.6$ and $e_3 = 0.1$.