Classification

Artificial Intelligence 1 A

# Task 1 - Data

The data that was given is not in an appropriate format for LibSVM. The first task is to convert the data sets into the format specified in the LibSVM ‘README’ file located in its root directory.

The source code for this task is located in ***appendix A***.

The result was tested with the checkdata.py using the command line "*python checkdata.py fileName*" and the results were "*no error*".

# Task 2- Normalisation

The data that was converted is raw data, it has not been normalised (scaled). Normalising the data is very important when using SVMs as it stops attributes with a large numeric range from dominating the classification model.

This task required the use of svm-scale.exe in the command prompt, with the need of the un-normalised data and the new file as parameters, as shown in ***appendix B***.

# Task 3 - Grid Search

When using the C-SVC SVM with the Gaussian radial basis kernel there are two tuneable parameters, C (cost) and γ (gamma). To achieve the highest classification rate possible it is very important to search for an optimal pair of these values. LibSVM makes this process very simple by including a Python script which carries out a grid search, a systematic search for optimal SVM parameters.

This task is to carry out a grid search on the training data set.

At first a general idea for this task was made by using the default values for C and γ using the command prompt - *python grid.py normalisedFile*, results seen in ***appendix C***.

Using the values of C and γ, and working out the log2 of each a finer search could be made using a longer command prompt - *python grid.py -log2c beg, end, step - log2g beg, end, step normalisedFile*, as seen in ***appendix D***.

The values of the first grid search for C was 2 to power 9, so for the finer search a +1/-1 of the power value was used as the beginning and end of the search.

# Task 4 - Classification

This task involves using LibSVM’s ‘svm-train’ and ‘svm-predict’, both command line applications. With the normalised training set as the input file, ‘svm-train’ can be used with the suitable parameter values discovered for c and γ during task 3(seen at the bottom of ***appendix E*** ). When the classification model is built the use of ‘svm-predict’ will be needed to classify the normalised testing set.

The command line of *svm-train.exe - g 0.0473661427034 - c 362.038671968 train.dat* was used first which outputted a .model file to be used by 'svm-predict'.

'svm-predict' used this file in conjunction with the normalised training set to create a predicted file, using the command line *svm-predict.exe test.dat train.dat.model train.dat*

As seen in ***appendix F***.

|  |  |  |
| --- | --- | --- |
| SVM | Number of misclassified instances | Accuracy rate % |
| Non-linear | 47 | 60.8333 |

# Task 5 - Classification Analysis

By using the output file of ‘svm-predict’ (generated during task 4), it is possible to map the classifications made back to the original instances. The order of the predictions in the output file is the same as the order of the instances in the testing data set.

|  |  |  |
| --- | --- | --- |
| SVM | Number of misclassified instances | Accuracy rate % |
| Non-linear | 47 | 80.83333333 |

|  |  |  |
| --- | --- | --- |
| Label | Actual label | Predicted label |
| Defective | 60 | 29 |
| Non-defective | 60 | 91 |

Could for this can be found in ***appendix G*** as well as the command prompt view in ***appendix H***.

As it can be seen by the output there was a total of 47 misclassified instances which was the same as task 4 but the only difference is that the accuracy between task 4 and 5 has a 20% difference.

This is because the features used for this test did not separate the classes were well.

# Task 6 - Linear Classification

When using the C-SVC SVM with the linear kernel there is only one tunable parameter, C (cost). In this experiment, it is required to use LibSVM’s ‘svm-train’ to train models using C = 10, C = 100, and C = 1000, in turn.

|  |  |  |  |
| --- | --- | --- | --- |
| SVM | C | γ | Accuracy rate % |
| Linear | 10 | - | 60.8333 |
| Linear | 100 | - | 60.8333 |
| Linear | 1000 | - | 61.6667 |

A look at the command prompt can be found in ***appendix I***.

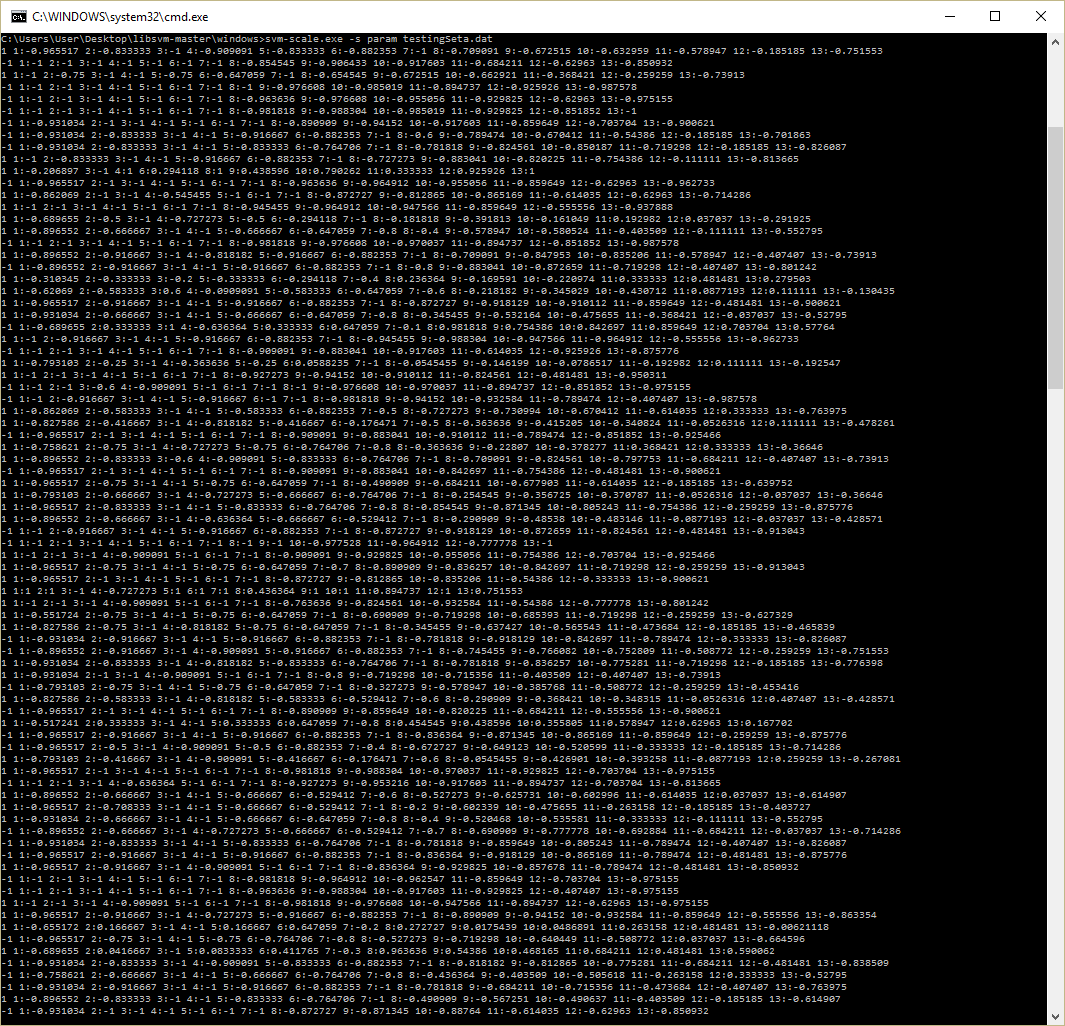
As shown by the table C = 1000 gives the best accuracy which is total different from task 4, as task 4 has the same accuracy as C = 10 and C = 100. Without a look at the full accuracy rate(full decimal) it is hard to determine if the accuracy is increasing with cost with just these three C values.

# Appendices

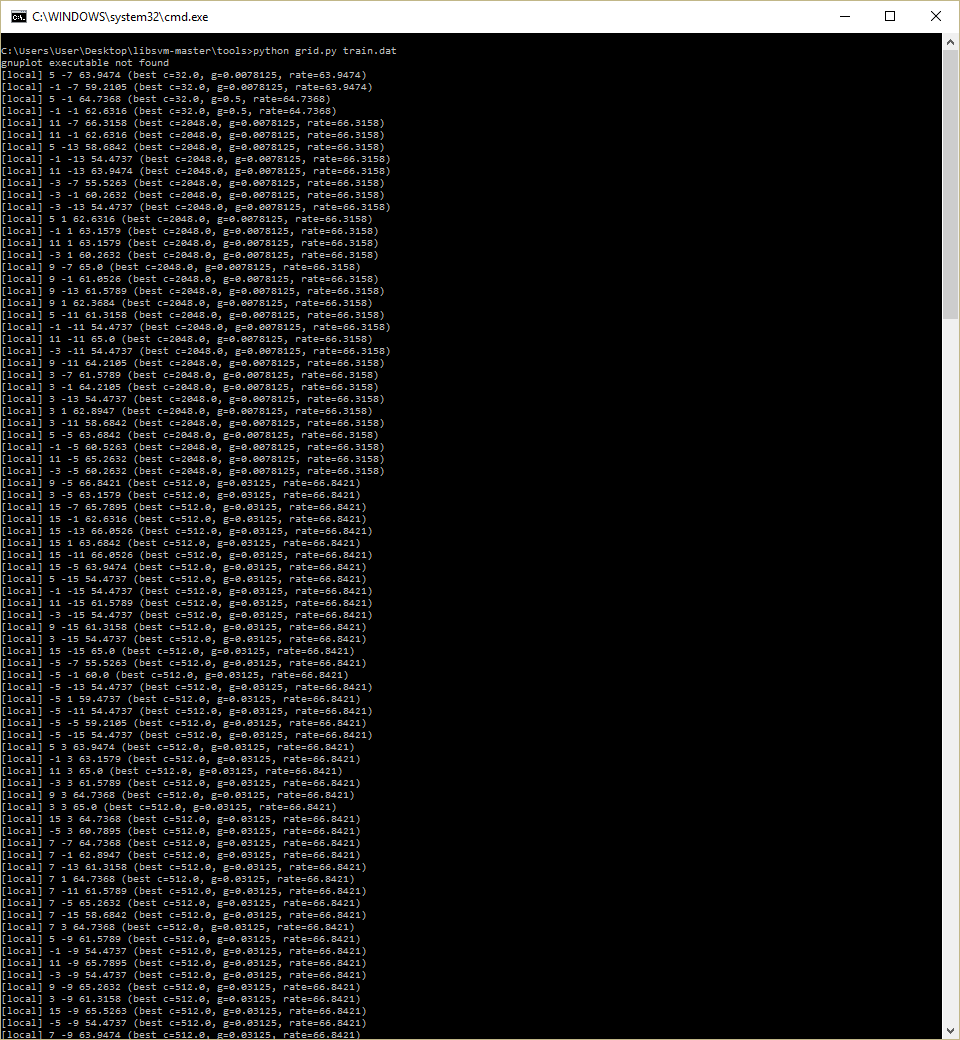
## Appendix A:

1. **import** csv
2. **import** sys
4. inputFile = sys.argv[1] #input file
5. outputFile = sys.argv[2] # output file
7. with open(inputFile, 'rb') as csvFile:
8. writeToFile = [] #list
9. myReader = csv.reader(csvFile, delimiter=',')#read from csv file
10. **for** row **in** myReader:
11. **if** row[len(row) - 1]== '1': # check character at end of line
12. newLine = "+" + row[len(row)- 1] # add new line with '+1' at front
13. **else**:
14. newLine = "" + row[len(row) - 1]# add new line with '-1' at front
15. **del** row[len(row) - 1]# delete it
16. count = 1
17. **for** v **in** row:
18. newLine = newLine + " " + str(count) + ":" + v # add contents to send line
19. count += 1
20. newLine = newLine + "\n"
21. writeToFile.append(newLine)
22. csvFile.close()# close input file
24. **del** writeToFile[0]# delete title
26. newFile = open(outputFile, "w")# create new file
27. **for** w **in** writeToFile:
28. newFile.write(w)#add contents of the list to the new file
30. newFile.close()# close output file

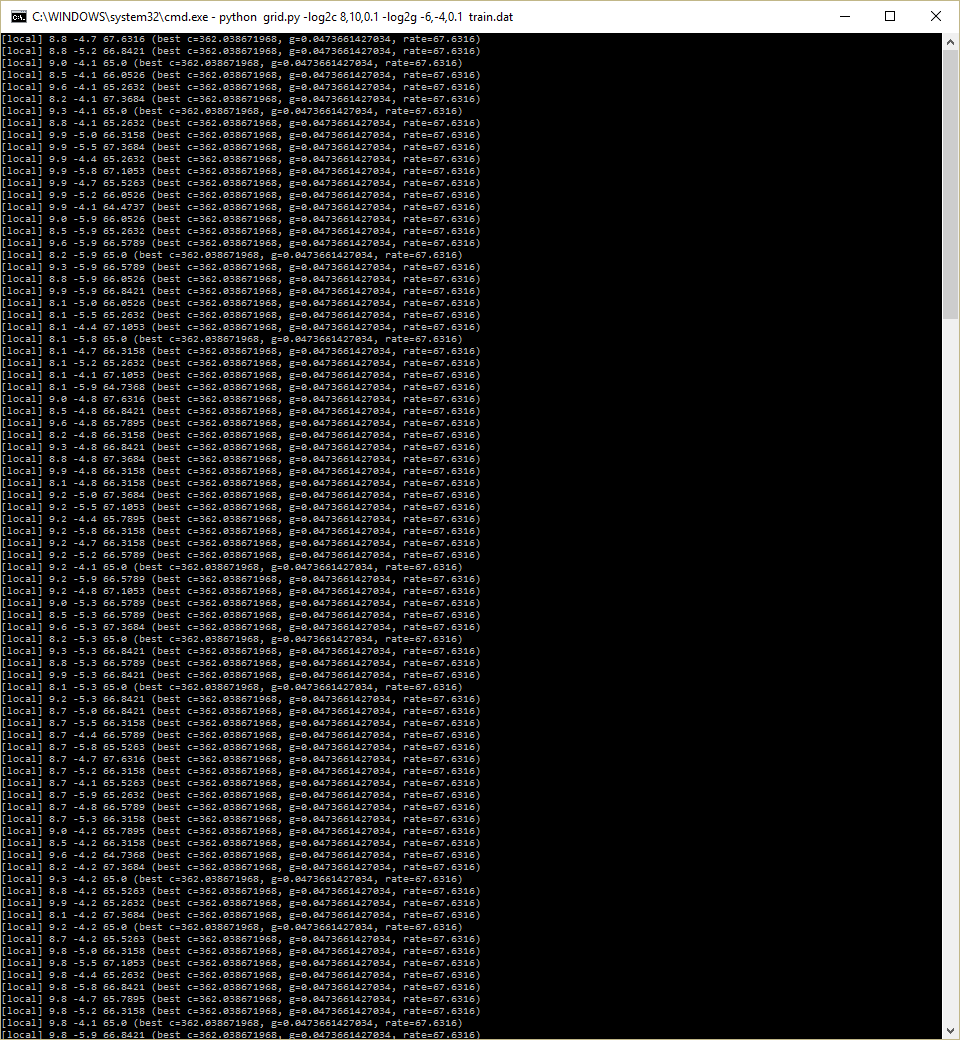
## Appendix B:



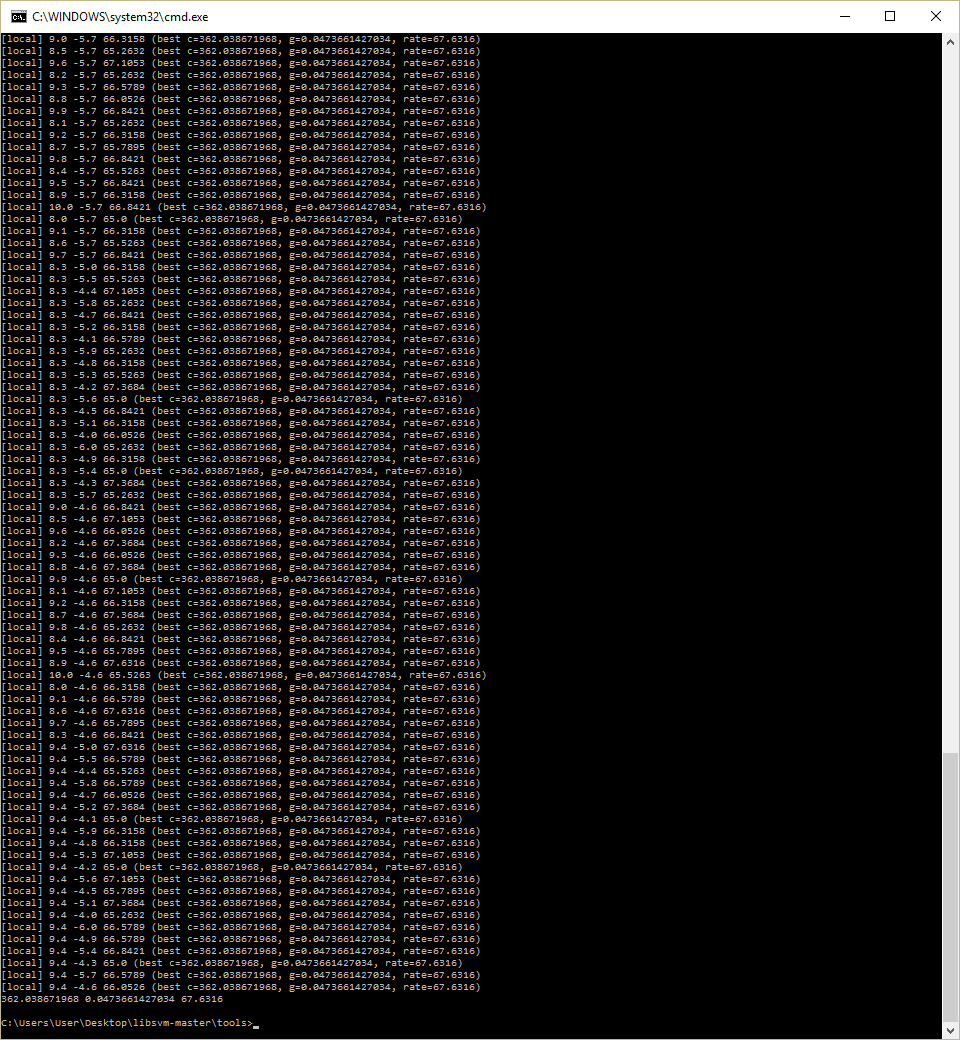
## Appendix C



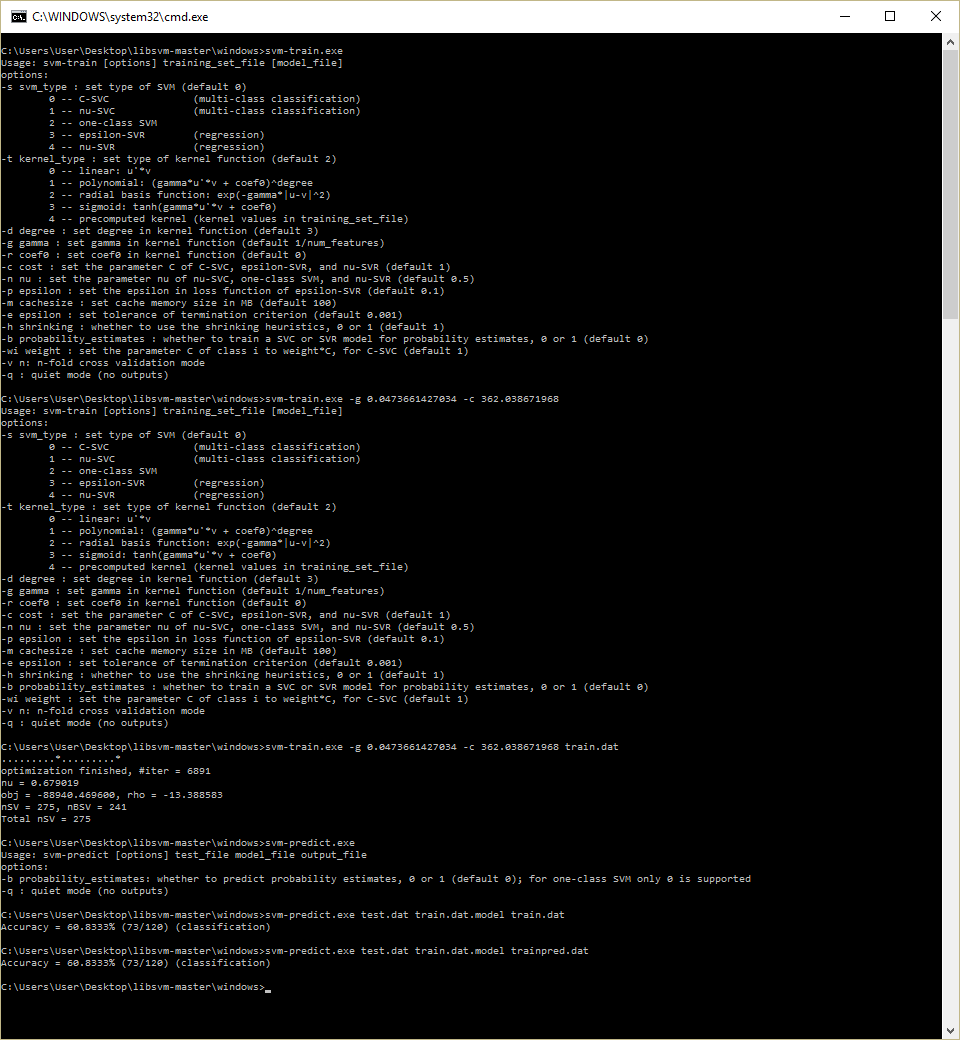
## Appendix D



## Appendix E



## Appendix F



## Appendix G

1. testFile = "test.dat"
2. trainFile = "trainpred.dat"
4. actual = []#list of acutal instances
5. predicted = []#list of predicted instances
6. #Combine files into a single list
7. with open (testFile) as textFile1:# open test file
8. with open(trainFile) as textFile2:# open predicted file
9. **for** e **in** range(0, 120): # look at all instances
10. line = textFile1.readline().rstrip('\n')#read line from test
12. actual.append(int(line[0:2]))
13. line = textFile2.readline().rstrip('\n')
14. predicted.append(int(line[0:2]))
15. textFile1.close()
16. textFile2.close()
18. #Analysis
19. incorrect = 0
20. incorrectly\_predicted\_nondefective = 0
21. incorrectly\_predicted\_defective = 0
22. act\_defective = 0
23. act\_nondefective = 0
24. pred\_defective = 0
25. pred\_nondefective = 0
27. **for** e **in** range(0,120):
28. original = actual[e]
29. classification = predicted[e]
31. **if** original == 1 **and** classification == -1:
32. incorrectly\_predicted\_nondefective +=1
34. **if** original == -1 **and** classification == 1:
35. incorrectly\_predicted\_defective += 1
37. **if** original == 1:
38. act\_defective += 1
40. **if** original == -1:
41. act\_nondefective += 1
43. **if** classification == 1:
44. pred\_defective += 1
46. **if** classification == -1:
47. pred\_nondefective += 1

50. incorrect = incorrectly\_predicted\_nondefective + incorrectly\_predicted\_defective
52. **print** "No of incorrect instances: " + str(incorrect)
53. **print** "No of instances that labeled as defective and incorrectly predicted as non-defective: " + str(incorrectly\_predicted\_nondefective)
54. **print** "No of instances that labeled as non-defective and incorrectly predicted as defective: " + str(incorrectly\_predicted\_defective)
55. **print** "No of actual defective: " + str(act\_defective)
56. **print** "No of actual non-defective: " + str(act\_nondefective)
57. **print** "No of predicted defective: " + str(pred\_defective)
58. **print** "No of predicted non-defective: " + str(pred\_nondefective)

## Appendix H



## Appendix I

