Grant Rosario

Document Classification Report 02/24

I started off the process by building the following program in order to build a document word matrix. Regular expressions were used to separated each document in the text file and NLTK was used to tokenize each document into a set of words.

```
import nltk
import string
import os
import numpy as np
import random
import re

train = open("doc_test.txt")
matrix = open("WD_matrix_test.txt", "w")

train = train.read()

doc_count = 0

labels = ["student", "project", "course", "faculty"]
matrix.write("Doc#" + "\t")
for label in labels:
    matrix.write(label + "\t")
matrix.write("\n")

docs = re.split('\n', train)
for doc in docs:
    doc_count += 1
    words = nltk.tokenize.word_tokenize(doc)
    student_in = 0
    course_in = 0
    project_in = 0
    faculty_in = 0
    for word in words:
        if word == "student":
            student_in = 1
            continue
        if word == "project":
            project_in = 1
            continue
        if word == "course":
            course_in = 1
            continue
        if word == "faculty":
            faculty_in = 1
            continue

    matrix.write('{}:\t{}\t{}\t{}\t{}\n'.format(doc_count, student_in, course_in, project_in, faculty_in))

    train.close()
    matrix.close()
```

My result was the following word matrix for the training and testing data text:

Doc#	student	project	course	faculty
1:	1	0	0	0
2:	1	0	0	0
3:	1	0	0	1
4:	1	0	0	0
5:	0	0	1	0
6:	0	0	0	1
7:	1	0	0	1
8:	0	0	0	1
9:	1	0	1	0
10:	0	1	1	0
11:	1	0	1	0
12:	0	0	1	1
13:	1	0	0	0
14:	1	0	0	0
15:	0	0	0	1
16:	0	0	1	1
17:	1	0	1	1
18:	1	0	1	0
19:	0	0	0	1
20:	1	0	0	0
21:	0	1	0	0
22:	0	0	1	0
23:	1	0	1	0
24:	0	0	0	1
25:	0	0	1	1
26:	1	0	0	0
27:	0	0	1	1
28:	0	1	0	0
29:	1	0	1	1
30:	0	0	0	1
31:	1	0	0	0
32:	1	0	0	0
33:	1	1	1	0
34:	1	0	0	0
35:	1	1	0	0
36:	0	0	1	0
37:	0	0	1	0
38:	1	0	0	0
39:	1	0	1	0
40:	1	1	1	0
41:	1	0	1	0
42:	1	0	1	0
43:	1	0	1	1
44:	1	0	1	0
45:	0	0	0	1
46:	1	0	1	1
47:	0	0	1	0
48:	1	1	0	0
49:	0	0	1	0
50:	0	1	0	0
51:	1	0	0	0
52:	0	1	0	0
53:	1	0	0	0
54:	1	0	0	0
55:	0	1	0	0
56:	1	0	1	1
57:	1	0	0	0
58:	1	0	0	0
59:	0	0	0	1
60:	1	1	0	0
61:	0	0	0	1
62:	0	0	1	1
63:	0	0	0	1

The next step was to write a program to re-format this matrix into the style which is compatible with LibSVM:

```
import nltk
import string
import os
import numpy as np
import random
import re

file = open("doc_train.txt")
matrix = open("libsvm_train.txt", "w")
full_txt = file.read()

#labels = ["student", "faculty", "project", "course"]
def writeToFile(class_num, class_name, class_in=0):
    matrix.write(class_num + " ")
    docs = re.split('\n', full_txt)
    doc_count = 0
    for doc in docs:
        doc_count += 1
        words = nltk.tokenize.word_tokenize(doc)
        class_in = 0
        for word in words:
            if word == class_name:
                class_in = 1
                matrix.write('{}:{} '.format(doc_count, class_in))
                break
            else:
                continue
    matrix.write('\n')

writeToFile("1", "student", 0)
writeToFile("2", "faculty", 0)
writeToFile("3", "project", 0)
writeToFile("4", "course", 0)
```

This resulted in the following type of reformatted file for the training data and testing data. These files were used as input for training and testing with LibSVM.

```
1 1:1 2:1 3:1 4:1 7:1 9:1 11:1 13:1 14:1 17:1 18:1 20:1 23:1 26:1 29:1 31:1 32:1 33:1 34:1 35:1 38:1 39:1
40:1 41:1 42:1 43:1 44:1 46:1 48:1 51:1 53:1 54:1 56:1 57:1 58:1 60:1 64:1 67:1 69:1 72:1 73:1 75:1 76:1 77:1
78:1 79:1 80:1 82:1 83:1 87:1 89:1 90:1 93:1 95:1 98:1 100:1 102:1 104:1 105:1 106:1 107:1 110:1 111:1 113:1
114:1 115:1 116:1 117:1 120:1 121:1 123:1 124:1 125:1 127:1 128:1 130:1 134:1 135:1 136:1 137:1 139:1 140:1
141:1 144:1 146:1 149:1 151:1 153:1 155:1 158:1 159:1 162:1 165:1 166:1 167:1 169:1 171:1 172:1 174:1 176:1
178:1 181:1 182:1 183:1 184:1 185:1 186:1 188:1 189:1 190:1 192:1 196:1 201:1 203:1 204:1 205:1 207:1 208:1
210:1 211:1 212:1 216:1 218:1 220:1 224:1 229:1 230:1 231:1 232:1 233:1 234:1 235:1 237:1 238:1 239:1 240:1
241:1 242:1 243:1 244:1 249:1 251:1 253:1 255:1 256:1 266:1 268:1 269:1 272:1 274:1 275:1 276:1 277:1 278:1
279:1 280:1 281:1 282:1 285:1 286:1 288:1 289:1 290:1 291:1 292:1 293:1 295:1 300:1 303:1 306:1 313:1 314:1
315:1 317:1 319:1 321:1 322:1 323:1 327:1 331:1 333:1 340:1 341:1 342:1 343:1 347:1 348:1 353:1 354:1 355:1
356:1 358:1 359:1 362:1 364:1 365:1 367:1 369:1 370:1 372:1 373:1 375:1 377:1 378:1 379:1 380:1 384:1 385:1
386:1 387:1 388:1 389:1 392:1 394:1 395:1 397:1 398:1 399:1 400:1 402:1 403:1 404:1 405:1 406:1 408:1 409:1
411:1 415:1 416:1 418:1 420:1 422:1 423:1 425:1 427:1 428:1 429:1 430:1 435:1 438:1 440:1 441:1 442:1 443:1
444:1 445:1 446:1 448:1 450:1 451:1 454:1 455:1 457:1 458:1 459:1 460:1 461:1 463:1 465:1 467:1 468:1 469:1
471:1 472:1 475:1 476:1 477:1 479:1 480:1 481:1 482:1 483:1 484:1 485:1 486:1 487:1 488:1 489:1 490:1 491:1
494:1 496:1 497:1 498:1 499:1 505:1 506:1 508:1 510:1 512:1 513:1 514:1 515:1 517:1 519:1 520:1 522:1 524:1
525:1 526:1 529:1 532:1 535:1 536:1 539:1 541:1 543:1 544:1 548:1 551:1 554:1 555:1 557:1 558:1 559:1 560:1
562:1 563:1 564:1 565:1 566:1 568:1 569:1 570:1 575:1 576:1 577:1 579:1 580:1 583:1 588:1 589:1 590:1 591:1
593:1 594:1 596:1 597:1 599:1 600:1 602:1 604:1 606:1 607:1 609:1 610:1 611:1 613:1 614:1 615:1 617:1 618:1
623:1 624:1 625:1 629:1 630:1 631:1 632:1 633:1 635:1 636:1 638:1 639:1 641:1 642:1 643:1 644:1 646:1 647:1
649:1 650:1 652:1 653:1 654:1 659:1 665:1 666:1 669:1 671:1 675:1 676:1 677:1 678:1 679:1 681:1 682:1 683:1
684:1 688:1 689:1 690:1 692:1 695:1 697:1 698:1 699:1 700:1 702:1 703:1 709:1 710:1 712:1 714:1 715:1 717:1
720:1 721:1 722:1 725:1 727:1 729:1 730:1 731:1 732:1 733:1 736:1 739:1 741:1 744:1 746:1 747:1 748:1 755:1
757:1 759:1 761:1 762:1 764:1 766:1 769:1 770:1 771:1 772:1 775:1 776:1 777:1 779:1 780:1 782:1 784:1 785:1
790:1 791:1 792:1 794:1 795:1 796:1 797:1 798:1 799:1 803:1 805:1 806:1 807:1 808:1 810:1 811:1 812:1 814:1
816:1 818:1 819:1 821:1 823:1 825:1 827:1 829:1 830:1 831:1 832:1 833:1 835:1 836:1 837:1 838:1 840:1 841:1
842:1 843:1 844:1 847:1 848:1 852:1 855:1 856:1 857:1 860:1 861:1 863:1 865:1 868:1 870:1 871:1 872:1 873:1
876:1 877:1 878:1 879:1 883:1 885:1 886:1 887:1 888:1 889:1 890:1 892:1 893:1 894:1 901:1 902:1 904:1 905:1
907:1 908:1 909:1 910:1 912:1 915:1 916:1 917:1 918:1 919:1 921:1 922:1 923:1 928:1 930:1 932:1 936:1 937:1
938:1 939:1 942:1 943:1 944:1 945:1 947:1 949:1 951:1 953:1 954:1 955:1 956:1 957:1 958:1 959:1 961:1 962:1
963:1 972:1 974:1 975:1 979:1 980:1 982:1 984:1 986:1 987:1 992:1 993:1 995:1 998:1 999:1 1000:1 1001:1
1002:1 1003:1 1008:1 1009:1 1010:1 1011:1 1013:1 1014:1 1017:1 1019:1 1020:1 1021:1 1022:1 1024:1 1025:1
1027:1 1028:1 1029:1 1030:1 1031:1 1034:1 1035:1 1037:1 1038:1 1039:1 1040:1 1041:1 1043:1 1045:1 1046:1
1048:1 1050:1 1051:1 1055:1 1056:1 1060:1 1062:1 1063:1 1065:1 1066:1 1067:1 1069:1 1070:1 1082:1 1083:1
1084:1 1086:1 1088:1 1090:1 1094:1 1095:1 1096:1 1098:1 1100:1 1101:1 1102:1 1103:1 1110:1 1112:1 1113:1
1115:1 1117:1 1119:1 1120:1 1125:1 1126:1 1129:1 1133:1 1135:1 1137:1 1138:1 1139:1 1141:1 1142:1 1143:1
1144:1 1145:1 1146:1 1147:1 1148:1 1152:1 1156:1 1159:1 1161:1 1162:1 1164:1 1165:1 1166:1 1168:1 1170:1
1171:1 1174:1 1175:1 1176:1 1178:1 1179:1 1182:1 1183:1 1185:1 1187:1 1190:1 1193:1 1197:1 1201:1 1202:1
1203:1 1204:1 1205:1 1206:1 1207:1 1208:1 1209:1 1211:1 1214:1 1218:1 1221:1 1223:1 1224:1 1225:1 1228:1
1230:1 1232:1 1235:1 1236:1 1237:1 1238:1 1240:1 1241:1 1248:1 1249:1 1250:1 1251:1 1253:1 1254:1 1255:1
1256:1 1257:1 1258:1 1259:1 1263:1 1265:1 1266:1 1267:1 1269:1 1270:1 1271:1 1272:1 1274:1 1278:1 1279:1
1280:1 1283:1 1285:1 1286:1 1287:1 1289:1 1291:1 1292:1 1293:1 1295:1 1296:1 1297:1 1298:1 1300:1 1301:1
1302:1 1304:1 1305:1 1306:1 1307:1 1308:1 1309:1 1312:1 1313:1 1316:1 1319:1 1320:1 1322:1 1323:1 1324:1
1329:1 1330:1 1331:1 1333:1 1334:1 1335:1 1336:1 1337:1 1338:1 1339:1 1341:1 1342:1 1343:1 1345:1 1346:1
1350:1 1353:1 1354:1 1357:1 1358:1 1360:1 1361:1 1364:1 1366:1 1367:1 1369:1 1370:1 1371:1 1374:1 1376:1
1377:1 1379:1 1380:1 1381:1 1382:1 1383:1 1384:1 1385:1 1386:1 1387:1 1389:1 1390:1 1392:1 1394:1 1395:1
1398:1 1399:1 1400:1 1404:1 1406:1 1408:1 1409:1 1410:1 1411:1 1412:1 1414:1 1416:1 1417:1 1419:1 1421:1
1422:1 1424:1 1425:1 1426:1 1430:1 1432:1 1433:1 1440:1 1441:1 1443:1 1444:1 1445:1 1447:1 1450:1 1451:1
1453:1 1455:1 1456:1 1457:1 1459:1 1461:1 1465:1 1466:1 1467:1 1472:1 1474:1 1475:1 1476:1 1477:1 1479:1
1482:1 1483:1 1484:1 1485:1 1486:1 1487:1 1488:1 1489:1 1490:1 1491:1 1492:1 1494:1 1495:1 1496:1 1502:1
1503:1 1504:1 1505:1 1507:1 1511:1 1513:1 1515:1 1516:1 1517:1 1518:1 1523:1 1525:1 1526:1 1527:1 1528:1
1529:1 1531:1 1534:1 1536:1 1537:1 1541:1 1543:1 1547:1 1550:1 1552:1 1553:1 1555:1 1556:1 1557:1 1560:1
1561:1 1563:1 1565:1 1567:1 1568:1 1569:1 1570:1 1571:1 1573:1 1575:1 1576:1 1578:1 1580:1 1582:1 1585:1
1586:1 1589:1 1590:1 1591:1 1592:1 1593:1 1594:1 1595:1 1596:1 1599:1 1603:1 1605:1 1606:1 1607:1 1611:1
1613:1 1614:1 1615:1 1617:1 1618:1 1619:1 1621:1 1623:1 1624:1 1625:1 1627:1 1628:1 1631:1 1632:1 1633:1
1635:1 1636:1 1637:1 1638:1 1639:1 1640:1 1642:1 1645:1 1647:1 1648:1 1649:1 1650:1 1655:1 1657:1 1659:1
1660:1 1662:1 1665:1 1667:1 1668:1 1669:1 1676:1 1677:1 1678:1 1679:1 1682:1 1684:1 1685:1 1686:1 1688:1
1689:1 1690:1 1691:1 1692:1 1693:1 1694:1 1695:1 1696:1 1699:1 1700:1 1701:1 1703:1 1705:1 1706:1 1708:1
```

The final step is to actually train and test a model with LibSVM. This is done with the following code:

```
import sys
import nltk
import string
import os
import numpy as np
import random
import re

#link to the svm and svmutil libraries in libvsm folder
sys.path.append('/Users/grantrosario/Downloads/libsvm-3.22/python' )
import svm
from svmutil import *

y, x = svm_read_problem('libsvm_train.txt')
model = svm_train(y, x, '-t 0')
y, x = svm_read_problem('libsvm_test.txt')
p_label, p_acc, p_val = svm_predict(y, x, model)
print("\n{}\n{}\n\n{}".format(p_label, p_acc, p_val))
```

Notice I am building a model using data from the training set. Then I am running a prediction to test the model on a new test data set. The results are on the next page.

```
grantrosario >> HW2 $ python prediction.py
*
optimization finished, #iter = 1
nu = 0.000981
obj = -0.000981, rho = 0.421285
nSV = 2, nBSV = 0
*
optimization finished, #iter = 1
nu = 0.001310
obj = -0.001310, rho = 0.359528
nSV = 2, nBSV = 0
*
optimization finished, #iter = 1
nu = 0.001118
obj = -0.001118, rho = 0.552823
nSV = 2, nBSV = 0
*
optimization finished, #iter = 1
nu = 0.001504
obj = -0.001504, rho = -0.233083
nSV = 2, nBSV = 0
*
optimization finished, #iter = 1
nu = 0.001460
obj = -0.001460, rho = 0.094891
nSV = 2, nBSV = 0
*
optimization finished, #iter = 1
nu = 0.001585
obj = -0.001585, rho = 0.348653
nSV = 2, nBSV = 0
Total nSV = 4
Accuracy = 25% (1/4) (classification)

[4.0, 4.0, 4.0, 4.0]
(25.0, 3.5, nan)

[[-0.19862677783227084, -0.1447282252783234, -0.25321408608160
99, 0.13834586466165416, -0.03503649635036504, -0.183835182250
39613], [-0.3104462972045121, -0.2730844793713162, -0.43767467
859139186, 0.16240601503759403, -0.10948905109489057, -0.29001
584786053874], [-0.25846002942618945, -0.21283562540929918, -0
.32811626607043043, 0.1518796992481203, -0.04379562043795626,
-0.20760697305863698], [-0.3271211378126534, -0.28356254092992
783, -0.43432084963666856, 0.1759398496240602, -0.080291970802
91975, -0.27258320126782876]]
```

ACCURACY = 25%

After analyzing the results and prediction code. I thought it was strange that the final accuracy of the prediction was so poor. I noticed that when LibSVM reads the data from the input files, it re-orders the indices and I believe this skews the result. I am not sure if this is normal behavior.