

Assignment 2

Text Classification With LibSVM

Introduction

The WebKB data set containing 2803 training instances and 1396 test instances is used in combination with LibSVM to conduct text classification. The data set consists of web pages collected from computer science departments of various universities. The web pages are labelled as student, faculty, project, or course. The data set was previously pre-processed with stemming and removal of stop words. The following report outlines all steps required to conduct text classification using LibSVM's terminal interface and the WebKB data set.

Data Formatting

In order to use the WebKB data set for classification with LibSVM, the data must first be converted to a format acceptable by LibSVM. The WebKB data is currently in the form of a text file, where each instance is separated by the new line character. Each instance consists of the instance's label followed by a sequence of stemmed words.

The LibSVM executables require that the data be in the form of:

label idx:value idx:value ... \n

Where every document is separated by new line, starts with the document's label, followed by a sequence of (index, value) pairs. For this assignment, TF-IDF values are used for the values. Therefore, the index refers to the terms index in the TF-IDF matrix, and the value refers to that terms TF-IDF value.

To achieve this format, a python script (prepare-data.py) was defined to read WebKB input files, calculate TF-IDF values, and write documents to output files in the form required by LibSVM.

First, the prepare-data python script reads the WebKB training and test data sets and constructs a list of all documents found in both training and test data. The data is combined prior to calculating TF-IDF scores to ensure that all vocabulary is included in the TF-IDF calculations. The length of training and test data sets is recorded, as it will be required to split the data back into two sets.

```
28 # combine training data and test data into single corpus
29 # allows generation of tf-idf matrix with complete dictionary
30 # stores lengths to reference when separating tf-idf matrix into test/train
31 allDocuments = []
32 fileDocumentCount = []
33 for file in inputFiles:
34     inputStream = open(file, 'r')
35     documents = inputStream.read().split('\n')
36     allDocuments += documents
37     inputStream.close()
38     fileDocumentCount.append(len(documents))
```

The class labels are removed from the documents and stored in a separate list, because they should not be included in the TF-IDF calculations and they will need to be referenced at a later time.

```
40
41 # separate class label from each document
42 # store class label in separate array
43 documentClasses = []
44 corpus = []
45 for document in allDocuments:
46     arr = document.split()
47     documentClass = arr.pop(0).strip()
48     documentClasses.append(classMap[documentClass])
49     corpus.append(' '.join(arr))
50
51
```

During construction of the documentClasses list, the document labels are mapped to integer values to comply with LibSVM format.

```
18
19 # define new class labels
20 classMap = {
21     "student": 1,
22     "faculty": 2,
23     "course": 3,
24     "project": 4
25 }
26
```

Next, Scikit-learn's TfidfVectorizer is used to construct the TF-IDF matrix using the complete data set (training and test data combined).

```
51
52 # create tf-idf matrix from the content
53 vectorizer = TfidfVectorizer()
54 tfidf_matrix = vectorizer.fit_transform(corpus)
55
56
```

Now that we have the document - TF-IDF matrix, we have all information required to begin writing the documents to their LibSVM compatible files.

A list of LibSVM formatted documents is created by combining the class label with the (index:tf-idf) pairs from the TF-IDF matrix.

```
56
57 # for every document
58 # construct string of the form (label idx:value idx:value idx:value)
59 formattedDocuments = []
60 for idx, row in enumerate(tfidf_matrix.toarray()):
61     result = str(documentClasses[idx])
62     for idx, value in enumerate(row):
63         if value > 0:
64             result += ' ' + str(idx) + ':' + str(value)
65     formattedDocuments.append(result)
66
```

The formattedDocuments list now contains a list of all documents, both test and training sets, in the form required by LibSVM. All that is left to do is to partition the formattedDocuments back into two sets, training data and test data.

```
67
68 # split formatted documents back into training / test data
69 trainingLength = formattedDocumentCount[0]
70 testLength = formattedDocumentCount[1]
71 formattedTrainingData = formattedDocuments[0:trainingLength]
72 formattedTestData = formattedDocuments[trainingLength:]
73
74
75 # write libsvm formatted training data to output file
76 directory = os.path.dirname(libsvmTrainingPath)
77 if not os.path.exists(directory):
78     os.makedirs(directory)
79 trainingStream = open(libsvmTrainingPath, 'w')
80 trainingStream.write('\n'.join(formattedTrainingData))
81 trainingStream.close()
82
```

Now that the data is partitioned back into the original training and test sets and written to output files, these files can be supplied to LibSVM to conduct classification.

Below is a preview of the LibSVM formatted data sets, train.txt and test.txt:

```
train.txt x
1 1 39:0.0417209828231 46:0.0414565758989 59:0.0510037292955 82:0.0355096916776 126:0.0322378487638 137:0.0393791541414 208:0.070393
2 1 284:0.399405363872 1339:0.225134442447 1391:0.0475696764422 1725:0.107705516845 1831:0.221015852513 1923:0.205166039111 2994:0.1
3 2 208:0.050677332539 335:0.0432930939513 347:0.0222762075129 441:0.0183697564211 504:0.0318090220554 517:0.0363577208647 612:0.16
4 1 72:0.233851837546 92:0.0907444907652 151:0.14856848478 208:0.0435557183456 226:0.0803459178707 495:0.32519543777 1109:0.08570405
5 4 352:0.0735635395483 492:0.075036374967 800:0.41773393447 904:0.0658013536742 990:0.0973304494171 1067:0.18794143162 1147:0.12010
6 2 445:0.149950713784 847:0.291432077908 904:0.162850074059 1317:0.15824379331 1391:0.061872351261 1564:0.292825142945 1667:0.22832
7 2 92:0.0698016406642 232:0.199240656676 492:0.0914621287113 774:0.0804223328326 1391:0.0609456823394 1548:0.131428098546 1837:0.04
8 2 1:0.0892725545212 40:0.0286599103526 44:0.0569821880892 92:0.0248244647303 307:0.0994167296718 401:0.0439595701643 411:0.0914061
9 1 192:0.33021943131 337:0.164020527187 904:0.151290023013 978:0.101847095692 1277:0.130230230891 1391:0.0574802897712 1461:0.18791
```

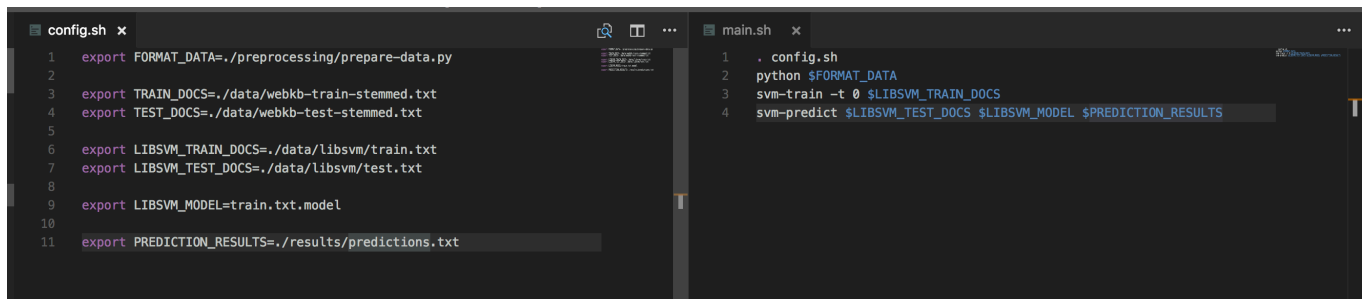
```
test.txt x
1 1 2302:0.59367709115 2693:0.444162797311 2778:0.306494446123 3169:0.217074801096 5214:0.236945973761 7153:0.365409444879 7559:0.34
2 3 440:0.0648089212362 441:0.0638353722151 1095:0.138168140926 1134:0.148688684284 1343:0.0742449636668 1391:0.106783191325 1885:0.0
3 1 192:0.0747950621242 346:0.164913996031 631:0.218913171348 873:0.0706304258533 961:0.108959671192 978:0.0692054354816 1067:0.0489
4 1 40:0.0506714315764 247:0.0999382825563 256:0.0714658999343 358:0.0584413810947 373:0.0637618898134 774:0.0505684073197 962:0.223
5 3 117:0.118184685309 208:0.0972614120876 371:0.108120912913 1299:0.200426820343 1370:0.120300964455 1391:0.0442316335939 1439:0.22
6 2 62:0.0495438214856 208:0.235925382687 402:0.0686237103995 924:0.210003202618 1099:0.0546464207602 1190:0.0664569588469 1391:0.17
7 3 80:0.134123716635 274:0.0712740525099 352:0.0430221832982 371:0.0357396291299 440:0.0354948385169 483:0.0685470458364 1033:0.060
8 1 69:0.055811151368 73:0.0255388840527 89:0.0283453045828 117:0.0253962217251 151:0.0356451129474 218:0.0523601966449 237:0.046946
9 2 1:0.163687743175 29:0.181300217119 92:0.0455174675916 315:0.141266183017 374:0.0445043023795 411:0.111733172247 445:0.0481589851
10 4 92:0.0688747983054 357:0.13734645964 978:0.213106820183 1244:0.073541917763 1343:0.250872470421 1391:0.0601364314473 1550:0.0796
11 1 389:0.120796949498 468:0.120289844055 469:0.106642702284 539:0.106378269679 634:0.120289844055 942:0.110098244642 961:0.07316139
12 2 14:0.0394443338652 62:0.0266719120459 73:0.025866728509 92:0.0220512471768 168:0.106763958193 207:0.0335653554217 208:0.042333680
13 1 92:0.277607318356 210:0.361683385541 359:0.252271840808 516:0.225605528189 1519:0.189496441837 1665:0.138732876878 2209:0.115924
14 3 82:0.0845358897356 205:0.26939642702 299:0.104357122848 349:0.0816046170817 363:0.116039357668 516:0.0945794989776 611:0.1247422
15 2 0:0.0729351536797 1:0.051158471818 33:0.0355995779788 72:0.146642549973 117:0.0331883017819 151:0.139745287117 257:0.04747768630
```

Text Classification with LibSVM Linear Kernel

Since LibSVM can be run from the terminal, a shell script (main.sh) has been written to automate all steps in the text classification process.

A config.sh (left) file was created to improve re-usability. The config.sh file defines the locations of the data formatting python script, the input WebKB data sets, paths to store the LibSVM-formatted files once converted, and the path to write the prediction results to.

The main.sh (right) file utilizes the config file to complete data formatting, model training, and test data evaluation. The entire process from data preparation to results is automated with the main.sh script.

A screenshot of a code editor with two tabs: 'config.sh' and 'main.sh'. The 'config.sh' tab is active and shows the following content:

```
1 export FORMAT_DATA=./preprocessing/prepare-data.py
2
3 export TRAIN_DOCS=./data/webkb-train-stemmed.txt
4 export TEST_DOCS=./data/webkb-test-stemmed.txt
5
6 export LIBSVM_TRAIN_DOCS=./data/libsvm/train.txt
7 export LIBSVM_TEST_DOCS=./data/libsvm/test.txt
8
9 export LIBSVM_MODEL=train.txt.model
10
11 export PREDICTION_RESULTS=./results/predictions.txt
```

The 'main.sh' tab is also visible and shows the following content:

```
1 . config.sh
2 python $FORMAT_DATA
3 svm-train -t 0 $LIBSVM_TRAIN_DOCS
4 svm-predict $LIBSVM_TEST_DOCS $LIBSVM_MODEL $PREDICTION_RESULTS
```

main.sh execution explanation:

1. Load configuration from config.sh
2. Execute python data formatting script, responsible for converting WebKB data into LibSVM format
3. Train Linear SVM model using LibSVM (-t 0 is required option for linear kernel)
4. Use model from 3 to make predictions on test data, writing results to output file and printing accuracy to console

Results

Execution of main.sh produces the following terminal output:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL
SB-FL-K4XCGTFM:LibSVM-Text-Classification justin.johnson$ ./main.sh
.*
optimization finished, #iter = 1932
nu = 0.319797
obj = -356.327548, rho = -0.621681
nSV = 946, nBSV = 303
.*
optimization finished, #iter = 1174
nu = 0.246612
obj = -204.021717, rho = -0.456594
nSV = 635, nBSV = 157
.*
optimization finished, #iter = 1108
nu = 0.152811
obj = -149.914116, rho = -0.600984
nSV = 554, nBSV = 98
.*
optimization finished, #iter = 1101
nu = 0.355507
obj = -228.144336, rho = 0.138362
nSV = 612, nBSV = 193
.*
optimization finished, #iter = 1009
nu = 0.180715
obj = -139.663172, rho = -0.160451
nSV = 488, nBSV = 81
*
optimization finished, #iter = 896
nu = 0.216699
obj = -113.868833, rho = -0.240060
nSV = 417, nBSV = 53
Total nSV = 1771
Accuracy = 90.616% (1265/1396) (classification)
SB-FL-K4XCGTFM:LibSVM-Text-Classification justin.johnson$
```

The LibSVM classifier with linear kernel correctly classified 90.616% of the test data instances. Additionally, the svm-predict executable wrote the predicted class values to the output file that was defined in config.sh. The labels are still in their integer format but can be easily converted back into the original string labels using the mapping defined earlier. A preview of first 16 predictions below:

```
predictions.txt x
1 1
2 3
3 1
4 1
5 3
6 2
7 3
8 1
9 2
10 4
11 1
12 2
13 1
14 3
15 2
16 3
```