Programming Assignment #4 (due on day 7 of Module 9)

Binary Text Classification

For this assignment you will work with the ***Systematic Review*** dataset that I described in the video lectures. A chief goal is to build a binary classifier using the training data and to use the trained model to make predictions on the test data. You will have considerable flexibility in which methods you use for the task, but you will be asked to investigate a few experimental conditions. The data for this assignment are formatted differently from the first three programs.

Data
The dataset is available as three tab separated value (TSV) files which can be found with the assignment in Canvas. Each line of the file consists of 10 tab delimited fields which are described below. There should be meaningful values for at least columns 1-3, but any of the other fields may be empty. The data are about 65 MB in size uncompressed and are encoded as UTF-8, although the data should be nearly if not entirely English. The data are split into three portions: ***train***, ***dev*** and ***test***. Train should be used to build your models. Dev should be used in your experiments. You will submit predictions for the Test partition.

Col.	Field	Description
1	Assessment	-1, 0, or 1; zero indicates unknown; 1=accept, -1=reject
2	Docid	Unique ID. Usually PubMed ID, or hashed ID
3	Title	Article title
4	Authors	List of authors (poss. separated with ';'). Unnormalized.
5	Journal	Journal title
6	ISSN	Numeric code for journal (possibly a list)
7	Year	Publication year
8	Language	Trigram for language code (e.g., "eng")
9	Abstract	Several sentence abstract from article
10	Keywords	List of keywords (formatting is highly variable) – should be ';' separated.

Download the data from the course web site and write a program that can process the data files and create feature vectors. You will then build a classifier from the training set. You can use any method for training (e.g., kNN, naïve Bayes, decision trees, SVMs, writing regular expression rules based on training samples, etc…). You may use, and in fact are strongly encouraged to use open source machine learning toolkits (e.g., NLTK, scikit-learn, spaCy, etc…). Links to a few such tools are available on my IR resource webpage. Many text classifiers are built using linear support vector machines. I describe using one SVM tool (SVMlight) below; it is available from https://svmlight.joachims.org/ (binaries are available for modern operating systems).

Note: these data are skewed towards the negative class. There are approximately 30 times as many negative examples as positive ones in the training data.

Baseline & Scoring (40 points)
Use features from only the title section (column 3) of the data. Make predictions against the Dev partition and report precision, recall and F_1 scores. Show the work in your computation (i.e., the numerators and denominators for precision and recall). Recall is the percentage of +1s in the Dev file that were correctly predicted to belong to the class; precision is the percentage of +1s in the output file (or that you predict are positive) which are indeed correct according to the Dev file labels. $F_1 = 2*P*R/(P+R)$.

Experiment #1: Is More Text Better? (20 points)
This time use features not only from the title, but also from the abstract and keywords fields. Report scores on the Dev partition as before.

Experiment #2: Surprise Me (20 points)
Conduct another experiment, preferably something non-trivial. Here are a few examples to consider:

- Tokenization: use stemmed words instead of plain words
- Tokenization: use character n-gram features

- Attributed fields: Have features be specific to the column they come from (i.e., Title:hospital is not the same as Abstract:hospital)
- Find other columns in the data that are helpful for prediction or determine that none are
- Try a different machine learning algorithm than your baseline approach
- Sweep parameters for your learning method, and find a setting that achieves the best results

You are not limited to the ideas above. Be sure to explain what your experimental condition is, and how the results compare to a model that uses "title + abstract + keywords" for features. Reported results should be on the Dev partition as before.

Test Set Predictions (20 points)
Using whatever approach you find to work best on the Dev set, produce predictions for the **Test** set. In addition to your PDF writeup, submit a plain text file that is named YOURJHEDID.txt with lines that are formatted as:

docid [tab] prediction (where docid is the string from Column 2 of the test data, and prediction is 1 or -1)

There should be no blank lines, no json, etc… Just plain text as described above. Your file should have just as many lines as the Test set and the docids in your predictions file should be in the same order. As a rough guideline, it is not too difficult to attain precision and recall of about 40% for this dataset.

Note: You are allowed to use open source toolkits for this assignment, and you do not need to implement algorithms (e.g., Naive Bayes) yourself. Just cite any toolkits used or any example code used or modified. No matter which classifier(s) you use, clearly describe your methods. For example, do you remove stopwords or perform stemming; do you use binary weights or TF/IDF weights; what tools do you use with which parameter settings; and how are features determined?

NLTK
NLTK is a Python-based machine learning toolkit that you might consider using. If you want to try it, you might consider reading through section 1.3 of this tutorial: https://www.nltk.org/book/ch06.html

SVMLight
SVMLight is another machine learner you may choose to use – executables are available for all modern operating systems. To use SVMlight you should process the data files, and write out files of vectors, one document vector per line. For example:

```
+1 5:1 13:1 78:1 … 15008:1
+1 5:1 45:1 78:1 15000:1
-1 3:1 13:1 87:1 12000:1
```

"+1" in the leftmost column indicates that the vector is positive for the class and "-1" indicates it is negative. Each *termid:value* element in this example uses binary weights; '1' indicates the binary presence of a term in the document, and terms not in the document (i.e., the zeros) are not written out. You do not have to use binary weights. For each vector the termids must be non-decreasing from left to right. After you create vectors, build a classifier and use it to make predictions for the dev/test partitions. Note: it is critically important that the termids are consistent in the training and dev/test data (e.g., 'disease' gets termid=37 for both the phase1.train and phase1.dev documents).

The example above is in the format SVMlight expects. To train a model with SVMlight:

```
% svm_learn –j 10 phase1.train phase1.mod        # "-j 10" addresses the class imbalance in this data
```

To run a test set against a trained model:

```
% svm_classify phase1.dev phase1.mod phase1.dev.out
```

The output file contains scores that are in line-by-line correspondence with the input vectors. Output scores (margins from the hyperplane) that are above zero indicate that the prediction is that the document belongs to the positive class.