Baseline & Scoring

Training data:

Total processed documents: 21700
Total positive documents: 700
Total negative documents: 21000

Dev data:

Total processed documents: 4850
Total positive documents: 150
Total negative documents: 4700

Column 3 (title) only prediction result:

Total processed documents: 4850
Total positive documents: 150
Total negative documents: 4700
Total predicted positive documents: 238
Total predicted negative documents: 4612
Actual negative and predicted negative (a): 4536
Actual negative and predicted positive (b): 164
Actual positive and predicted negative (c): 76
Actual positive and predicted positive (d): 74

```
P= d/(b+d);
R= d/(c+d);
F= 2*P*R/(P+R);
```

Precision: 0.31092436974789917
Recall: 0.49333333333333335
F1 score: 0.38144329896907214

Accuracy on test set: 95.05% (4610 correct, 240 incorrect, 4850 total)
Precision/recall on test set: 31.09%/49.33%

Comments:

For this assignment I use SVMLight as machine learner. I spent some time to figurine out how to provide training data, and start with +1 documents (700) with less -1 documents (700), total training data is 1400 documents. I tried to provide even documents to train the model and assumed it can provide better prediction, but unfortunate the result is not precise enough. So the I later on decided to use the all data set with “-j 10” option as training data.

Experiment #1: Is Longer Better?

Column 3 (title)+ column 9 (abstracts) + column10 (keywords) result:

Total processed documents: 4850
Total positive documents: 150
Total negative documents: 4700
Total predicted positive documents: 142
Total predicted negative documents: 4708
Actual negative and predicted negative (a): 4626
Actual negative and predicted positive (b): 74
Actual positive and predicted negative (c): 82
Actual positive and predicted positive (d): 68

```
P= d/(b+d);
R= d/(c+d);
F= 2*P*R/(P+R);
```

Precision: 0.4788732394366197
Recall: 0.4533333333333333
F1 score: 0.4657534246575343

Accuracy on test set: 96.78% (4694 correct, 156 incorrect, 4850 total)
Precision/recall on test set: 47.89%/45.33%

Comments:

For some reason the program I wrote takes too much time to process data for col3+col9+col10. (each document takes 3 seconds to parse data, so total up to 20 hrs to process 21700 training documents, and I did it but somehow I can only parse 14000 documents) The result shows longer words does improve recall and precision. It make sense since we provide more data to train the model compared to with less training set.

Experiment #2: Surprise Me

Column 3 (title) with stem method and term frequency

Total processed documents: 4850
Total positive documents: 150
Total negative documents: 4700
Total predicted positive documents: 325
Total predicted negative documents: 4525
Actual negative and predicted negative (a): 4460
Actual negative and predicted positive (b): 240
Actual positive and predicted negative (c): 65
Actual positive and predicted positive (d): 85

```
P= d/(b+d);
R= d/(c+d);
F= 2*P*R/(P+R);
```

```
Precision: 0.26153846153846155
Recall: 0.5666666666666667
F1 score: 0.3578947368421053

Accuracy on test set: 93.71% (4545 correct, 305 incorrect, 4850 total)
Precision/recall on test set: 26.15%/56.67%
```

Column 10 (keywords) with stem method

```
Total processed documents: 4850
Total positive documents: 150
Total negative documents: 4700
Total predicted positive documents: 335
Total predicted negative documents: 4515
Actual negative and predicted negative (a): 4440
Actual negative and predicted positive (b): 260
Actual positive and predicted negative (c): 75
Actual positive and predicted positive (d): 75

Precision: 0.22388059701492538
Recall: 0.5
F1 score: 0.30927835051546393

Accuracy on test set: 93.09% (4515 correct, 335 incorrect, 4850 total)
Precision/recall on test set: 22.39%/50.00%
```

Comments:

Basically I choose term weight instead of binary weight to train the model. I didn't choose tf-idf since I feel each document is relative short and guess the impact should be similar as simply term weight. I tried column 3 (title) with stemmed/term weight. Use stem should improve result as first impression.

It turns out better recall rate but lower precision rate. My thought is mainly due to title 3 might not relevant enough as training data so how to improve data is not likely show significant better result.

Without further experiments, I guess column 10 itself should be precise enough because it's mainly human manually selected keywords, should perform the best result since sometimes title and abstract content might distract the training model. The result turns out even worse col3 title only, which surprise me. I would think title only for training model is a very effective way rather than with longer data, since it's short and good result.

Test Set Predictions:

9ef84cbb-2f69-3bdc-872f-b986823fe4cd	-1
ee1f7198-f1cf-3ee9-a794-ad7d5c3165be	-1
4a24e3aa-ebc1-3826-b98c-302815763ede	-1
e57a42d3-db98-3b5b-b7ad-5b2efb73f76e	-1
e2a94bd0-10e9-3ca1-a12f-18705ef15b74	-1
70b79896-03fb-3140-ba90-6e6982d3f2b4	-1
ad63deb0-c068-3772-b8bf-006291b14c49	-1
ab2a3500-29f3-3319-9966-aaa5833e4cd1	-1
9ecd76da-eda8-3bea-9906-6f32c71ecb59	-1

……

Txt file sent as separate email. Total processed documents: 4814, I use col3 only as training model.

Source Code:

```java
// -----------------------------------------------------------------
// Chin-Ting Ko
//
// 605.744 Information Retrieval - Spring 2015 PA4
// -----------------------------------------------------------------

import java.io.*;
import java.util.*;

public class Scanner {
    String filename;
    BufferedReader reader = null;
    Dict dict = new Dict();

    public Scanner(String fname) throws IOException {
        filename = fname;
        reader = new BufferedReader(new InputStreamReader(new FileInputStream(filename)), 65536);
    }

    public void terminate() throws IOException {
        reader.close();
    }

    // Process input file line by line
    public void process() throws IOException {
        String line = null;
        while (true) {
            line = reader.readLine();
            //System.out.println(line);
            if (line == null) {           // Have reached EOF
                break;
            } else {
                processLine(line);
            }
        }
    }
    int docNum = -1;
    StringBuffer buf = null;

    public void processLine(String line) {
        if (line.length() < 1) { return; }
            if (line.contains("hash")) {      // start of doc
                // reset state
               buf = new StringBuffer();
               buf.append(line);
               processTextForDoc();
                //docNum = extractDocNum(line); // might check line is well-formed
            } else {
               return;
                //buf.append("\n");
            }
        }

  int numDocs = 0;

    public void processTextForDoc() {
        if (buf != null) {
            numDocs++;
            // do stuff with buf and docnum (add to dictionary or inverted file)
            String text = buf.toString();
            StringTokenizer tok = new StringTokenizer(text, "/.?!,;()\"' \t\n");
            Document doc = new Document();
            while (tok.hasMoreTokens()) {
                String token = tok.nextToken().toLowerCase();
                StringBuffer tokenStem = new StringBuffer(token);
                String tokenStemmed = tokenStem.toString();

                if (tokenStem.length()>4){
                tokenStemmed = tokenStem.delete(5, tokenStem.length()).toString();
                }

                if (tokenStemmed.length() > 1 && !tokenStemmed.equals("-1") && !
tokenStemmed.contains("hash:") &&! stopwords.contains(tokenStemmed)) {// remove stop words

                //if (token.length() > 1) {
                    doc.addTerm(tokenStemmed);
                    //System.out.println(tokenStemmed);
                }
            }
            dict.handleDoc(docNum, doc); // Update counts for words in this document
        }
    }

    // Very short stopword list
    static HashSet stopwords = new HashSet();
    static {
        String [] swords = {"the", "of", "to", "and", "a", "in", "for", "is", "that", "on",
"with", "said", "by", "it", "as", "be", "are", "at", "from", "not", "was", "or", "an", "this",
"will", "he", "have", "has", "but", "its", "new", "were", "they", "would", "more", "can", "we",
"his", "other", "no", "been", "about", "also", "their", "you", "than", "all", "up", "if", "may",
"had", "there", "such", "after", "some", "into", "any", "out", "do", "under", "use", "only",
"these", "last", "am", "over", "so", "because", "most", "could", "each", "used", "says",
"support", "through", "between", "high", "did"};
```

```
        // which, many, where, how, who, when, what
        for(int i=0; i<swords.length; i++) {
            stopwords.add(swords[i]);
        }
    }


    public static void main (String [] args) throws IOException {

        Scanner scanner = new Scanner(args[0]);
        PrintWriter output = new PrintWriter (new FileWriter("parseData.txt"));
        //PrintWriter output2 = new PrintWriter (new FileWriter("+1output.txt"));
        //PrintWriter output3 = new PrintWriter (new FileWriter("-1output.txt"));
        PrintWriter output4 = new PrintWriter (new FileWriter("prediction.txt"));

        scanner.process();
        scanner.terminate();
        System.out.println(scanner.numDocs + " docs processed");
        System.out.println(scanner.dict.size() + " unique terms observed");
        System.out.println(scanner.dict.numCounts() + " total terms encountered");
        System.out.println();

        //scanner.dict.printTermid(100,150);
        scanner.dict.writeIF();
        //scanner.dict.writeDict();
        //scanner.dict.loadDict();
        System.out.println();
        int numDocs = 0;
        int positiveDoc=0;
        int negativeDoc=0;
        int predictPositiveDoc=0;
        int predictNegativeDoc=0;
        double a=0; //actual negative and predicted negative
        double b=0; //actual negative and predicted positive
        double c=0; //actual positive and predicted negative
        double d=0; //actual positive and predicted positive
        double F=0; // F1 score
        double P=0; //precision
        double R=0; //recall

        BufferedReader reader = null;
        reader = new BufferedReader(new InputStreamReader(new FileInputStream(args[1])), 65536);

        BufferedReader reader2 = null;
        reader2 = new BufferedReader(new InputStreamReader(new FileInputStream(args[2])), 65536);

        String line = "0";
        String line2 = "0";

        while (true) {
               if (line!=null){
        line = reader.readLine();
        line2 = reader2.readLine();
        //System.out.println(line);}
               if (line==null){break;}


        if (line.startsWith("-1")){
```

```
				//System.out.print("-1 ");
				output.print("-1 ");
				negativeDoc++;
				//output3.println(line);

			}
			if (line.startsWith("1")){
				//System.out.print("+1 ");
				output.print("+1 ");
				positiveDoc++;
				//output2.println(line);
			}
			if (line.startsWith("0")){
				//System.out.print("0 ");
				output.print("0 ");
			}

			StringBuffer buf = new StringBuffer();
			TreeMap<Integer, Integer> termidSorted = new TreeMap();
			String column1= line.substring(0,line.indexOf('	'));
			//System.out.println("Assessment: "+column1);
			int actual=Integer.parseInt(column1);

			String column2to9= line.substring(line.indexOf('	')+1);
			//System.out.println(column2to9);
			String column2= column2to9.substring(5,column2to9.indexOf(' '));
			//System.out.println("Docid: "+column2);
			output4.print(column2+"	");

			double prediction=Double.parseDouble(line2);
			if (prediction>0){
				predictPositiveDoc++;
			   output4.println("+1");
			}
			if (prediction<0){
				predictNegativeDoc++;
			    output4.println("-1");
			 }
			if (actual<0 && prediction <0){
				a++;
			}
			if (actual<0 && prediction >0){
				b++;
			}
			if (actual>0 && prediction <0){
				c++;
			}
			if (actual>0 && prediction >0){
				d++;
			}

			String column3to9= column2to9.substring(column2to9.indexOf('	')+1);
			//System.out.println(column3to9);
			String column3= column3to9.substring(0,column3to9.indexOf(' '));
			//System.out.println("Title: "+column3);

			String column10= line.substring(line.lastIndexOf('	')+1);
			String column1to9= line.substring(0,line.lastIndexOf('	')-1);
```

```
        String column9= column1to9.substring(column1to9.lastIndexOf('	')+1);
        //System.out.println("Abstract: "+column9);
        //System.out.println("Keywords: "+column10);

        buf.append(column3);
        //buf.append(column9);
        //buf.append(column10);

        if (buf != null) {
               numDocs++;
               System.out.println("working on doc: "+numDocs);
            String text = buf.toString();
            StringTokenizer tok = new StringTokenizer(text, "/.?!,;()\"' \t\n");
            Document doc = new Document();
            while (tok.hasMoreTokens()) {
                String token = tok.nextToken().toLowerCase();
                StringBuffer tokenStem = new StringBuffer(token);
                String tokenStemmed = tokenStem.toString();

                if (tokenStem.length()>4){
                tokenStemmed = tokenStem.delete(5, tokenStem.length()).toString();
                } //5-stemming for token stemming

                if (tokenStemmed.length() > 1 && !tokenStemmed.equals("-1") && !
tokenStemmed.contains("hash:") &&! stopwords.contains(tokenStemmed)) {// remove stop words

                      LexPayload payload = scanner.dict.loadPostings(tokenStemmed);

                    /*if (payload != null) {
                        System.out.print(tokenStemmed+"  Term id: " + payload.termid+":");
                        System.out.print("  Occurence: " + 1+" ");
                        //System.out.print("  OccurenceF: " + payload.postings.get(1));
                    } else {
                        System.out.println(tokenStemmed+"  Error: missing term");
                    }
                    */
                    if (payload != null&& payload.termid!=0 ) {
                      //termidSorted.put(payload.termid, payload.occurrence); //use binary
weight
                      termidSorted.put(payload.termid, payload.postings.get(1)); //use term
weight

                        //System.out.print(payload.termid+":"+payload.occurrence+" ");
                        //output.print(payload.termid+":"+payload.occurrence+" ");
                        //System.out.print(payload.termid+":"+payload.postings.get(1)+" ");
                    } else {
                        //System.out.println(tokenStemmed+"  Error: missing term");
                    }
                }                 //output.print(termidSorted);
            }

            Set set = termidSorted.entrySet();
            // Get an iterator
            Iterator i = set.iterator();
            // Display elements
            while(i.hasNext()) {
               Map.Entry termidSorted_vector = (Map.Entry)i.next();
```

```
              output.print(termidSorted_vector.getKey() + ":"+termidSorted_vector.getValue()+"
");
            }
            output.println();
        }

      }

    }
        System.out.println("Total processed documents: "+numDocs);
        System.out.println("Total positive documents: "+positiveDoc);
        System.out.println("Total negative documents: "+negativeDoc);
        System.out.println("Total predicted positive documents: "+predictPositiveDoc);
        System.out.println("Total predicted negative documents: "+predictNegativeDoc);
        System.out.println("Actual negative and predicted negative (a): "+(int)a);
        System.out.println("Actual negative and predicted positive (b): "+(int)b);
        System.out.println("Actual positive and predicted negative (c): "+(int)c);
        System.out.println("Actual positive and predicted positive (d): "+(int)d);

        P= d/(b+d);
        R= d/(c+d);
        if(P+R!=0){
        F= 2*P*R/(P+R);}

        System.out.println("Precision: "+ P);
        System.out.println("Recall: "+ R);
        System.out.println("F1 score: "+ F);


        reader.close();
        output.close();
        //output2.close();
        //output3.close();
        output4.close();
    }
}
```