Programming Project Evaluation Form

Student Name:	**Project:**	**Date:**
Correctness Criteria: • Reads normal input data • Handles incorrect input data • Calculates correct results • Outputs results properly • Prints appropriate error messages		**Score:______/ 50**
Design Criteria: • Problem decomposition • Choice of data structures • Choice of algorithms • Program efficiency (space/time)		**Score:______/ 20**
Documentation Criteria: • Comments for classes and methods • Comments describing algorithms • Comments describing data structures • Description of program design • Description of test results • Description of known problems		**Score:______/ 10**
Style Criteria: • Method and variable names • Program indenting • Use of white space • Ordering of methods • Easy to read code		**Score:______/ 10**
Testing Criteria: • Normal input data • Incorrect input data • Special cases for data structures • Special cases for algorithms		**Score:______/ 10**
Grader Comments:		**Total:______/ 100**

1. Objective

To disambiguate word pairs using a Naive-Bayesian technique and answer to some questions.

2. Installation

Programming language: Python

Source code location: /home/sbillah/nlp2/

Corpus Selection: I select "**senseval**" corpus, which is specially designed for WSD. Here is the download link: **http://www.senseval.org/.**

3. Design (Naive Bayesian Approach)

3.1 Algorithm:

I use two in-memory Hash-tables to store conditional probabilities of contexts associated with the pseudowords.

```
// preprocess corpus files.
for each sense-file f:
     1. replace the word with pseudoword and remember its sense in
        <tag> region.

     2. extract the context words in both side of the pseudoword, and
     store the contexts in two files: training, and testing by 8:2
     ratio.

//training
For each training file f:
     for each line in f:
          1. update C(context-words), C(word), & sense sk in respected
             hash-tables.
     from the counts, compute $P(c_i|s_k)$, $p(s_k)$and store in hash-tables.

//testing
for each testing file f:
     for each line in f:
          1. apply Laplace smoothing on conditional probabilities.
          2. compute argmax score($s_k$) using the formula in the book.
          3. compare the predicted value with actual value.

return accuracy in percentage.
```

Time Complexity:

- **Preprocessing phase:** O(# of lines containing word1) + O(#of lines containing word2)
- **Training phase:** 2*O(2*context_size * #lines in training file)
- **Testing phase:** O(2*context_size * #lines in testing file)
- **Overall:** O(2*context_size*(#of word1+ #of word2)

Space Complexity:

- **Overall:** O(2 * context_size * (#of word1 + #of word2))

3.2 Corpus Description:

The **Senseval** WSD corpus has total 35 sense-tagged words. Each word has more than 5 senses. But due to the simplified requirement of our homework, I ignore all those senses. Therefore, for a word pair, I consider only two senses (0,1). Here are my selected word-pairs and their individual occurrence in the corpus.

Pair	Words	Word Counts
1	amaze	319
	behaviour	1003
2	sack	296
	sanciton	101
3	knee	477
	onion	29
4	accident	1303
	wooden	370

Below is a snapshot of some lines from "***accident.cor***" file (context for word, 'accident'):

```
800001
Late on Thursday night it was travelling at about three metres a second in
wind blowing at 20 to 25 knots when an empty car fell off just as it reached
the top.
The <tag "532675">accident</> appeared to have little effect on the Christmas
party, except to lengthen it considerably.

800002
An image of earnest Greenery is almost tangible.
Eighteen years ago she lost one of her six children in an <tag
"532675">accident</> on Stratford Road, a tragedy which has become a pawn in
the pitiless point-scoring of small-town vindictiveness.
```

```
800003
It's a sentiment I recommend to you all.
The <tag "532675">accident</> occurred on the Saturday of the annual Popular
Flying Association (PFA) rally at Cranfield.

...
```

3.3 Context Selection:

I varied context length from 1 to 19 (on both side) as shown in the figure below:

Context size
word
Context size

Different level of accuracy is obtained under different context size. The results are given in the next chapter.

3.4 Laplace Smoothing:

During testing phase, some context-words are not seen before in training phase. Instead of assigning zero probability for them, I use Laplace Smoothing. The Laplace Smoothing is given below:

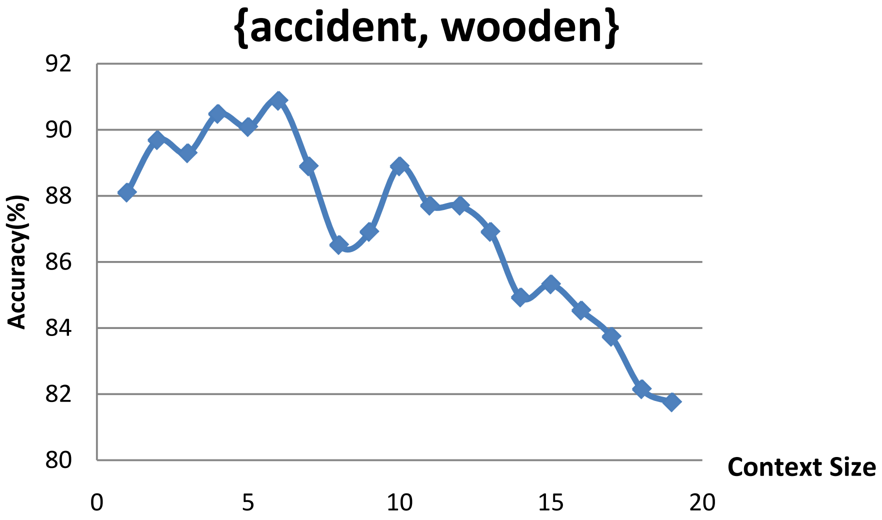
- If $P(ci|s_1) > 0$:
  - o $P(ci|s_1)' = (P(ci|s_1)*10000+1)/(10000 + context\_size)$
- Else:
  - o $P(ci|s_1)' = 1.0/(10000 + context\_size)$

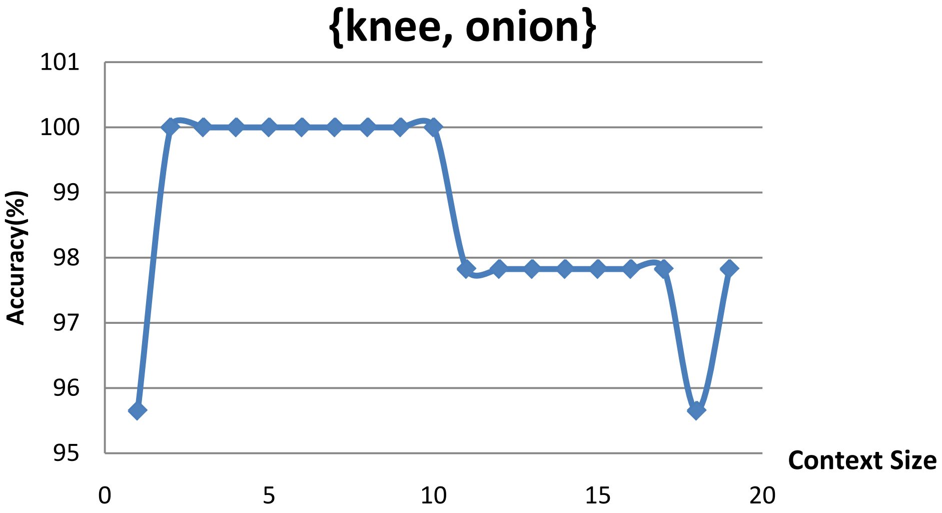
4 Experimental Result

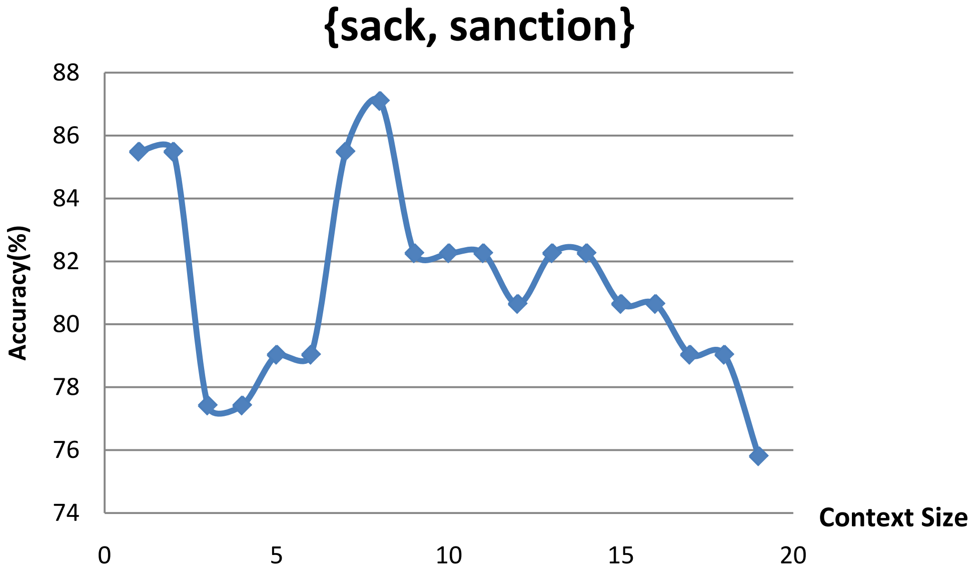
Here, I provide the experimental results for different context size and word-pairs. After the table, I also present these data graphically for better understanding.

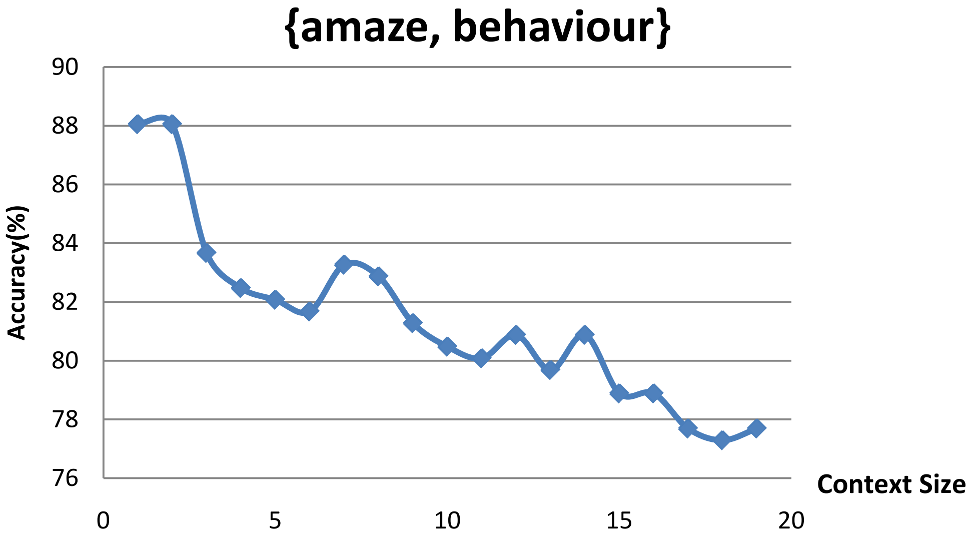
Context Size	Pair 1 accuracy {accident, wooden}	Pair 2 accuracy {knee, onion}	Pair 3 accuracy {sack, sanction}	Pair 4 accuracy {amaze, behaviour}
1	88.09524	95.65217	85.48387	88.04781
2	89.68254	100	85.48387	88.04781
3	89.28571	100	77.41935	83.66534
4	90.47619	100	77.41935	82.47012
5	90.07937	100	79.03226	82.07171
6	90.87302	100	79.03226	81.67331
7	88.88889	100	85.48387	83.26693
8	86.50794	100	87.09677	82.86853
9	86.90476	100	82.25806	81.2749

10	88.88889	100	82.25806	80.47809
11	87.69841	97.82609	82.25806	80.07968
12	87.69841	97.82609	80.64516	80.87649
13	86.90476	97.82609	82.25806	79.68127
14	84.92063	97.82609	82.25806	80.87649
15	85.31746	97.82609	80.64516	78.88446
16	84.52381	97.82609	80.64516	78.88446
17	83.73016	97.82609	79.03226	77.68924
18	82.14286	95.65217	79.03226	77.29084
19	81.74603	97.82609	75.80645	77.68924









Discussion:

1. Accuracy always decrease with the increase of context size.
2. For different word-pairs, maximum accuracy is obtained in different context size.
3. The overall performance of Naive Bayesian disambiguation is around 90%.

Q&A

2.9 Relative frequency:

File: `DavidBowie.html`
Entropy: `3.52728218653`

1	2370
0	1909
3	885
2	1502
5	784
4	750
7	881
6	721
9	1358

8	917
a	19075
c	8595
b	4576
e	23268
d	8301
g	3754
f	5294
i	20024

h	7057
k	3306
j	423
m	4891
l	11101
o	11614
n	13074
q	144
p	7083

s	14664
r	12658
u	4556
t	16047
w	5067
v	2178
y	2469
x	1038
z	232

File: `Genghis Khan - Wikipedia, the free encyclopedia.html`
Entropy: `3.55870437955`

1	1844
0	1870
3	873
2	1562
5	686
4	566
7	505
6	610
9	624

8	594
a	20479
c	7362
b	3328
e	21399
d	7111
g	6384
f	4749
i	20236

h	9353
k	4675
j	848
m	5568
l	11457
o	11204
n	14745
q	263
p	6122

s	13442
r	11698
u	4415
t	15763
w	4265
v	1788
y	2359
x	1218
z	392

File: `Steve Jobs - Wikipedia, the free encyclopedia.html`
Entropy: `3.61251296863`

1	7046
0	7798
3	1564
2	6815
5	1808
4	1605
7	1667
6	1966
9	2120

8	1811
a	36239
c	18155
b	7692
e	47051
d	11058
g	6116
f	10656
i	31138

h	14120
k	4429
j	2588
m	9194
l	22943
o	23755
n	26050
q	179
p	16747

s	31064
r	23907
u	6540
t	32956
w	9403
v	5120
y	4506
x	3221
z	504

File: Winston Churchill - Wikipedia, the free encyclopedia.html
Entropy: 3.54645118781

1	6843	8	1769	H	21543	s	34129
0	5721	a	45163	K	8332	r	35857
3	1761	c	21142	J	1441	u	10682
2	3863	b	9324	M	12329	t	43826
5	2560	e	56370	L	34877	w	12176
4	1703	d	19968	O	29943	v	5852
7	1273	g	9962	N	34098	y	7013
6	1797	f	13635	Q	445	x	3895
9	2795	i	49316	P	16278	z	654

Finally, the total Corpus frequency:

1	18103	8	5091	H	52073	s	93299
0	17298	a	120956	K	20742	r	84120
3	5083	c	55254	J	5300	u	26193
2	13742	b	24920	M	31982	t	108592
5	5838	e	148088	L	80378	w	30911
4	4624	d	46438	O	76516	v	14938
7	4326	g	26216	N	87967	y	16347
6	5094	f	34334	Q	1031	x	9372
9	6897	i	120714	P	46230	z	1782

2.10 KL Divergence

KL-divergence <1,2>	0.003688
KL-divergence <2,1>	0.003688
KL-divergence <1,3>	**0.03306**
KL-divergence <3,1>	**0.03306**
KL-divergence <2,3>	0.028468
KL-divergence <3,2>	0.028468

So, the corpus 1(english1) and corpus 3 (french1) have the highest score. In fact, these two corpus are same and translation of each other, which justifies the result.

```python
'''
Created on Mar 13, 2013

@author: Masum
'''

import re
from math import log
from collections import defaultdict
import os
import sys

def replace_tag2(line_text, tag, pseudoword, line_no, context_size=2):  # <tag "532675">
    header = str(line_no) + ":" + pseudoword + ":" + str(tag) + "\n";
    line_text = re.sub('<tag "\d+">\s*\w+</>', pseudoword, line_text, 1)
    line_text = re.sub('[.|"|,|;|:|!|(|)|?|`|\']', '', line_text);
    parsed = [tok.lower() for tok in line_text.split(' ') if len(tok)>2];
    contexts = []

    index = 0;
    try:
        index = parsed.index(pseudoword);
    except:
        #print line_text+'\n'
        return None;

    #add left tokens
    if (index>=context_size):
        contexts.extend(parsed[index-context_size : index])
    #add right tokens
    if len(parsed)>(index+context_size):
        contexts.extend( parsed[index+1:index+1+context_size])
    #print contexts
    line_text = header + (' ').join(contexts) + "\n"
    return line_text;

def process_file(file_in, max_lines, file_out, pseudoword, sense, start_line, context_size):
    mode = 'w+' if (start_line==0) else 'a';
    f_train = open(file_out+".train.txt", mode);
    f_test = open(file_out+".test.txt", mode);
    curr_line = 0;
    tokenized_line="";

    with open(file_in, 'r') as f:
        line_text = "";
        within_a_line = False
        for line in f:
            if re.match("\n", line):
                # if start_line>3:break;
                within_a_line = False;
                tokenized_line = replace_tag2(line_text, sense, pseudoword, start_line, context_size)
                if tokenized_line:
                    if curr_line<max_lines*.8:
                        f_train.write(tokenized_line);
                    else:
                        f_test.write(tokenized_line);
                    start_line += 1;
                curr_line+=1;
                line_text = ""
                continue;

            if re.match('\d+', line):
                within_a_line = True;
                continue;

            if within_a_line:
                line_text += line.strip() + " ";
    f_train.close();
    f_test.close();
    return start_line;
# end

def build_corpus(file_in1, line_no1,file_in2, line_no2,file_out, pseudoword, contex_size):
    # 1st file
    line_no = process_file(file_in1, line_no1,file_out, pseudoword , 0, 0, contex_size);
    # 2nd file
    process_file(file_in2, line_no2,file_out, pseudoword , 1, line_no, contex_size);
# end;


def run_traning(file_in, pseudoword):
    hashes = [defaultdict(float), defaultdict(float)]
    sense_types= [0,1];
    count_senses=[0.0, 0.0];
    current_sense = -1

    with open(file_in, 'r') as f:
        line_no=0;
        for line in f:
            #if line_no>3:break;
            m = re.match("\d+:"+pseudoword+":(\d)", line) #0:accidentwooden:0
            if m:
                current_sense = int(m.group(1))
                count_senses[current_sense]+=1;
                line_no += 1;
            else:
                for context in  line.strip().split(" "):
                    hashes[current_sense][context]+=1;
        #priors
        priors = [1.0*count_senses[0]/(count_senses[0]+count_senses[1]), 1.0*count_senses[1]/(count_senses[0]+count_senses[1])];

        #conditionals
        for sense in sense_types:
            for context in hashes[sense]:
                hashes[sense][context] = 1.0*hashes[sense][context]/count_senses[sense];

    #print hashes;
    return sense_types, priors,hashes ;


def run_disambiguation(file_train, file_test, pseudoword, context_size):
    sense_types, priors, conditionals =  run_traning(file_train, pseudoword);
    actual_sense = -1;
    predicted_sense = -1;
    scores  = [0.0, 0.0];

    #print "priors: ", priors;
    #print "sense types: ",sense_types;
    #print "training instances: ",count_senses
    #print conditionals[1];

    factor = 10000;
    count_corrects=0;
    count_wrong= 0;

    with open(file_test, 'r') as f:
        line_no=0;
        for line in f:
            m = re.match("\d+:"+pseudoword+":(\d)\n", line) #0:accidentwooden:0
            if m:
                actual_sense = int(m.group(1))
                line_no += 1;
            else:
                contexts = line.strip().split(" ");
                for sense in sense_types:
                    scores[sense] = log(priors[sense]);

                    for context in contexts:
                        if context in conditionals[sense]:
                            prob = conditionals[sense][context]
                            scores[sense] += log((prob*factor+1.0)/(factor+context_size));
                        else:
                            scores[sense] += log(1.0/factor);
                        #end inner for
                    #end sense
                #end outer for
                if scores[0]>scores[1]:
                    predicted_sense = 0;
                else:
                    predicted_sense = 1;

                #calculate accuracy
                if predicted_sense==actual_sense:
                    count_corrects+=1;
                else:
                    count_wrong+=1;
            #print actual_sense, predicted_sense;
        #print 'accuracy: ', count_corrects*100.0/(count_corrects+count_wrong), '\n';
        print context_size, ' ',count_corrects*100.0/(count_corrects+count_wrong);


def run_pair(w1, line_no1, w2, line_no2, context_count):
    build_corpus('wsd/'+w1+'.cor', line_no1,'wsd/'+w2+'.cor', line_no2,'wsd/'+w1+w2+'.bag', w1[2:]+w2[2:], context_count)
    run_disambiguation('wsd/'+w1+w2+'.bag.train.txt', 'wsd/'+w1+w2+'.bag.test.txt',w1[2:]+w2[2:], context_count);


def hw3():
    for i in range(1,20):
        #print 'for run', i,':';
        run_pair('4.amaze', 319,'4.behaviour', 1003, i);
        #run_pair('3.sack', 296,'3.sanction', 101, i);
        #run_pair('2.knee', 477,'2.onion', 29, i);
        #run_pair('1.accident', 1303,'1.wooden', 370, i);
#end


def get_file_chars(fname):
    alphabet = ['a','b','c','d','e','f','g','h','i','j','k','l','m','n','o','p','q','r','s','t','u','v','w','x','y','z','0','1','2','3','4','5','6','7','8','9'];
    alpha_dict = defaultdict(float);

    total=0
    with open(fname, 'r') as f:
        for line in f:
            for c in line.lower():
                total+=1;
                if c in alphabet:
                    alpha_dict[c] += 1.0;
    return alpha_dict;
#end

def entropy(alpha_dict):
    info = 0.0
    total = 0. + sum(alpha_dict.values())
    for c in alpha_dict:
        #print c, ' ', alpha_dict[c]
        p = alpha_dict[c]/total;
        info += -p*log(p, 2);
    print 'entropy is: ',info;

def kldiv(_s, _t):
    if (len(_s) == 0):return 1e33
    if (len(_t) == 0):return 1e33

    ssum = 0. + sum(_s.values())
    tsum = 0. + sum(_t.values())

    vocabdiff = set(_s.keys()).difference(set(_t.keys()))
    lenvocabdiff = len(vocabdiff)

    """ epsilon """
    epsilon = min(min(_s.values())/ssum, min(_t.values())/tsum) * 0.001

    """ gamma """
    gamma = 1 - lenvocabdiff * epsilon

    """ Check if distribution probabilities sum to 1"""
    sc = sum([v/ssum for v in _s.itervalues()])
    st = sum([v/tsum for v in _t.itervalues()])

    if sc < 9e-6:
        print "Sum P: %e, Sum Q: %e" % (sc, st)
        print "*** ERROR: sc does not sum up to 1. Bailing out .."
        sys.exit(2)
    if st < 9e-6:
        print "Sum P: %e, Sum Q: %e" % (sc, st)
        print "*** ERROR: st does not sum up to 1. Bailing out .."
        sys.exit(2)

    div = 0.
    for t, v in _s.iteritems():
        pts = v / ssum
        ptt = epsilon
        if t in _t:
            ptt = gamma * (_t[t] / tsum)
        ckl = (pts - ptt) * log(pts / ptt)
        div +=  ckl
    return div
#end of kldiv



def get_corpus_count(corpus_path):
    total_freq = defaultdict(float);

    for filename in os.listdir (corpus_path):
        #print filename
        file_count = get_file_chars(os.path.abspath(corpus_path)+os.path.sep+filename);
        for k in file_count:
            total_freq[k] += file_count[k];
        #print '\n'
    #get_file_chars("kl/English1/test.txt");
    #for k in total_freq:
        #print k, ' ', total_freq[k];
    return total_freq


corpus_path1 = "kl/English1/"
corpus_path2 = "kl/English2/"
corpus_path3 = "kl/French1/"

print "KL-divergence <1,2>:", kldiv(get_corpus_count(corpus_path1),get_corpus_count(corpus_path2));
print "KL-divergence <2,1>:", kldiv(get_corpus_count(corpus_path2),get_corpus_count(corpus_path1));

print "KL-divergence <1,3>:", kldiv(get_corpus_count(corpus_path1),get_corpus_count(corpus_path3));
print "KL-divergence <3,1>:", kldiv(get_corpus_count(corpus_path3),get_corpus_count(corpus_path1));

print "KL-divergence <2,3>:", kldiv(get_corpus_count(corpus_path2),get_corpus_count(corpus_path3));
print "KL-divergence <3,2>:", kldiv(get_corpus_count(corpus_path3),get_corpus_count(corpus_path2));
```