Homework 5: Lexicons and Distributional Semantics

This is due on **Friday, 11/10 (11pm)**

How to do this problem set

Most of these questions require writing Python code and computing results, and the rest of them have textual answers. Write all the textual answers in this document, show the output of your experiment in this document, and implement the functions in the `python` files.

Submit a PDF of thie .ipynb to Gradescope, and the .ipynb and all python files to Moodle.

The assignment has two parts:

- In the first part, you will experiment with Turney's method to find word polarities in a twitter dataset, given some positive and negative seed words.
- In the second part, you will experiment with distributional and vector semantics.

Your Name: <>

List collaborators, and how you collaborated, here: (see our grading and policies page (http://people.cs.umass.edu/~brenocon/inlp2017/grading.html) for details on our collaboration policy).

- *name 1*

In [1]:
```
%load_ext autoreload
%autoreload 2
```

In [2]:
```
# Initiallizing seed list
pos_seed_list = ["good", "nice", "love", "excellent", "fortunate", "correct",
"superior"]
neg_seed_list = ["bad", "nasty", "poor", "hate", "unfortunate", "wrong", "infe
rior"]
```

Part 1: Lexicon semantics

Recall that PMI of a pair of words, is defined as:

$$PMI(x,y) = log\frac{P(x,y)}{P(x)P(y)}$$

The Turney method defines a word's polarity as:

$$Polarity(word) = PMI(word, positive\_word) - PMI(word, negative\_word)$$

where the joint probability $P(w, v)$ or, more specifically, $P(w\ NEAR\ v)$ is the probability of both being "near" each other. We'll work with twe ets, so it means: if you choose a tweet at random, what's the chance it contains both `w` and `v`?

(If you look at the Turney method as explained in the SLP3 chapter, the "hits" function is a count of web pages that contain at least one instance of the two words occurring near each other.)

The positive_word and negative_word terms are initially constructed by hand. For example: we might start with single positive word ('excellent') and a single negative word ('bad'). We can also have list of positive words ('excellent', 'perfect', 'love', ...) and list of negative words ('bad', 'hate', 'filthy',....)

If we're using a seed list of multiple terms, just combine them into a single symbol, e.g. all the positive seed words get rewritten to POS_WORD (and similarly for NEG_WORD). This $P(w, POS\_WORD)$ effectively means the co-ocurrence of w with any of the terms in the list.

For this assignment, we will use twitter dataset which has 349378 tweets. These tweets are in the file named `tweets.txt`. These are the tweets of one day and filtered such that each tweet contains at least one of the seed words we've selected.

Question 1 (15 points)

The file `tweets.txt` contains around 349,378 tweets with one tweet per line. It is a random sample of public tweets, which we tokenized with twokenize.py's tokenizeRawTweetText() (https://github.com/myleott/ark-twokenize-py/blob/master/twokenize.py)). The text you see has a space between each token so you can just use `.split()` if you want. We also filtered tweets to ones that included at least one term from one of these seed lists:

- Positive seed list: ["good", "nice", "love", "excellent", "fortunate", "correct", "superior"]
- Negative seed list: ["bad", "nasty", "poor", "hate", "unfortunate", "wrong", "inferior"]

Each tweet contains at least one positive or negative seed word. Take a look at the file (e.g. `less' andgrep`'). Implement the Turney's method to calculate polarity scores of all words.

Some things to keep in mind:

- Ignore the seed words (i.e. don't calculate the polarity of the seed words).
- You may want to ignore the polarity of words beignning with `@` or `#`.

We recommend that you write code in a python file, but it's up to you.

QUESTION: You'll have to do something to handle zeros in the PMI equation. Please explain your and justify your decision about this.

textual answer here We have to handle zero in two places in the PMI equation, first in the denominator part of the equation, because $1/0$ is $infinity$ and $log(infinity)$ is undefined. Also, we have to manage zero in the numerator because $log(0)$ is undefined as well. Here I am adding hyperparameter alpha in the PMI equation which is 1 if not defined something else by the user. So, by adding one to numerator and denominator it will make sure that the equation never have to face zero and if the counts are not zero or the PMI does not have to handle zero, 1 is divided by the large number in this case N (number of tweets) and it will not effect the calculation.

Question 2 (5 points)

Print the top 50 most-positive words (i.e. inferred positive words) and the 50 most-negative words.

Many of the words won't make sense. Comment on at least two that do make sense, and at least two that don't. Why do you think these are happening with this dataset and method?

In [3]:
```
from helper import *
import operator
tweets = readfile('tweets.txt')
positive_negative_word, word_counts = count(tweets, pos_seed_list, neg_seed_li
st)
#result = polarity(tweets, positive_count, negative_count, word_counts)
```

```
The len of the tweets.txt 174689
```

In [4]:
```
pol = polarity(tweets, positive_negative_word, word_counts, 1.0)
```

In [5]:

```
import operator
most_positive = sorted(pol.items(), key=operator.itemgetter(1), reverse=True)
most_negative = sorted(pol.items(), key=operator.itemgetter(1))
count = 0
print "******50 Most Positive Words*******"
for word,val in most_positive:
    print word, val
    count+=1
    if count > 50:
        break

count = 0

print "\n********50 Most Negative Words**********"
for word,val in most_negative:
    print word, val
    count+=1
    if count > 50:
        break
```

```
******50 Most Positive Words*******
Hello 11.0919143693
evening 10.8681847088
Birthday 10.5744836465
ʔ10.1568442365
👍 9.73556708457
♡ 9.10267909553
❤ 8.86277886853
March 8.61008247481
craze 8.58598835231
Happy 8.47378796666
Thanks 8.39456779064
23 8.35409200989
lovely 8.25532031711
USATour2017 8.20827702712
AlwaysJaDine 8.20105677914
birthday 8.10131383339
thank 8.07709194188
kindness 7.98896801146
Thank 7.97577312632
😘 7.94104327357
Lauren 7.86954285832
\n\nPanaloMOTO 7.80775900809
Congrats 7.80404567297
London 7.7327116277
😍❤ 7.7260841315
BIRTHDAY 7.61599344146
First 7.57808326914
:D 7.43313294637
makeup 7.41415800878
Morning 7.38961189488
ol 7.33721161848
😊 7.13152587805
Team 7.12273295005
movie 7.07586073155
HAPPY 7.06616259857
hood 7.06242077942
wonderful 7.02614021438
goodnight 6.96858227023
xx 6.91960826563
flowers 6.9117698173
💕 6.80351749099
practice 6.77195160545
collection 6.76437591993
Check 6.76385925756
thanks 6.74669315928
ji 6.71726784195
they'd 6.70364607372
nd 6.65910892777
🙏 6.64659288428
shall 6.64057735244
Justin 6.62197413259

********50 Most Negative Words**********
crimes -11.3414343952
closure -9.67914113146
suggesting -9.67009129594
```

```
violence -9.00793928311
they\ -8.81182440418
presidency -8.66442511978
claims -8.60327965229
moods -8.58078620993
disability -8.46003944554
cigarettes -8.45698600805
violent -8.18848675502
Obamacare -8.16311407584
un… -8.11949388353
families -8.09240208182
explain -8.08934659007
sick -8.06172233069
despicable -7.97104122482
Pelosi -7.94660389977
POC -7.87573104501
vandalism -7.73383140272
habit -7.71402877542
victims -7.69340948822
blaming -7.65704184405
Saf… -7.64211619383
threats -7.55437727952
Obama’s -7.48577012344
Crime -7.46870221493
Worse -7.46399632389
diary\ -7.44654941029
gonna… -7.44654941029
designed -7.41494407087
JCCs -7.39567707991
Victims -7.37182586409
generals -7.30659744986
centers -7.30036690011
botc… -7.27978619241
passed -7.1888144142
raid -7.08653556508
leak -7.07722334879
crime -7.0439525327
https://t.co/ELGdDnERnv -6.98005960034
Latino -6.95944031314
responsibility -6.92567345067
condemn -6.88933074695
Oh -6.88589053246
annual -6.85716146402
DeVos -6.81016056355
anti-Semitism -6.80789041501
justify -6.80789041501
anti-Semitic -6.79824811752
JCC -6.78954127634
```

Textual answer here.

The Word♡, Thanks, Birthday in the Positive words make sense also, words like condemn, crime and violance is making sense in the negative words.

These are the words which defines the positiveness or negativeness of the data, for example If we are wishing birthday to someone:-

```
"Wishing you happy birthday"
```

Also The word Condemn being negative makes sense, given example:

```
"I condemn the unfortuniate terorist attacks in the humanity"
```

The words like nd, they are ending up in the positive words and words like familes, gonna they are not making sense.

The word like nd are more neutral than negative, for example the If I am using the wishing someone birthday wish I may use nd,

```
"Wishing you very very happy birthday nd have a blast today"
```

or another example of nd can be

```
  "I wish you were never born mr p. you are such shame nd unforunate for this worl
d"
```

Also, Word like families and gonna are again more neutral than negative like

```
"This families on vacations will the best this gonna happen this break"
```

This problem is arrising because of the words which are less in frequency and have possibility of being in one class than other. So, this words like 'nd' is appering in the most negative class than the positive class.

Question 3 (5 points)

Now filter out all the words which have total count < 500, and then print top 50 polarity words and bottom 50 polarity words.

Choose some of the words from both the sets of 50 words you got above which accoording to you make sense. Again please note, you will find many words which don't make sense. Do you think these results are better than the results you got in Question-1? Explain why.

In [6]:
```
# Write code to print words here
count = 0
print "\n******* Most Positive Words after filetering *******\n"
for word,val in most_positive:
    if word_counts[word] >= 500:
        print word, val
        count+=1
    if count > 50:
        break

count = 0
print "\n******* Most Negative Words after filetering *******\n"
for word,val in most_negative:
    if word_counts[word] >= 500:
        print word, val
        count+=1
    if count > 50:
        break
```

```
******* Most Positive Words after filetering *******

? 10.1568442365
❤ 8.86277886853
Happy 8.47378796666
Thanks 8.39456779064
birthday 8.10131383339
thank 8.07709194188
Thank 7.97577312632
ol 7.33721161848
😊 7.13152587805
movie 7.07586073155
hood 7.06242077942
thanks 6.74669315928
Von 6.45513884392
Be 6.38227196063
Taylor 6.10532106971
soon 6.0919510928
follow 6.026388402
morning 5.94340750638
Teyana 5.92851632067
😍 5.79106129583
Keef 5.76199166336
https://t.co/I46MhN5tag 5.76199166336
Herban 5.76199166336
https://t.co/inU4PQoS7z 5.7481346287
😂😂😂😍 5.7481346287
happy 5.59693919737
Always 5.54769646434
!! 5.42125628289
best 5.40545499507
Have 5.32223187847
amazing 5.30010990617
beautiful 5.29740428689
:) 5.16042751999
far 4.96262048409
speech 4.84964465446
together 4.80478848749
heard 4.71328843313
today 4.69827760769
Decay 4.66452550662
night 4.60012997205
work 4.59827345801
miss 4.58587175315
God 4.56653210251
friends 4.55107175138
luck 4.52914376234
! 4.50622093762
pregnancy 4.46952366465
learn 4.37325879806
giving 4.35788535187
fall 4.35565704008
President 4.33882887758

******* Most Negative Words after filetering *******

crimes -11.3414343952
```

```
cigarettes -8.45698600805
explain -8.08934659007
Saf… -7.64211619383
crime -7.0439525327
Oh -6.88589053246
CHILD -6.54350390516
ugly -6.52962938954
store -6.44339274539
used -6.31611098992
condemning -6.26759436894
gets -5.81953486801
marijuana -5.76941648508
feel -5.7603551075
stands -5.68549759768
woman -5.56288709735
evil -5.52007207225
ME -5.45721648501
hell -5.36928161882
women -5.36209593953
something -5.22521113894
When -5.0624679646
its -5.02411321777
want -4.89041293301
sorry -4.82779032053
ppl -4.80728352087
make -4.7668013538
sad -4.70387895519
united -4.58612472876
Dems -4.56259663505
Democrats -4.55903363901
😭 -4.46456309607
Why -4.36733110629
bc -4.12388925509
school -4.10779613239
when -3.92662307735
after -3.8970257752
food -3.84593083754
shit -3.80818716277
without -3.76495982826
won't -3.72918710162
hurt -3.71460240841
against -3.67638245493
ask -3.67458047847
away -3.67312896468
ass -3.64857538743
idea -3.59431030118
getting -3.55046173756
there -3.51561523659
going -3.48690288272
w -3.45892921957
```

Textual answer here.

Yes, this results look better than the previous results. for example if we see the first word of the most positive word now is a♡than hello as a previous case. Hello is more neutral word, it can be used to both positive and negative conversations "Hello mister please mind your own bussiness."

This is happining because when we are filtering out words which have frequency less than 500, This are the words which might have appeared in the one class than other.

Question 4 (5 points)

Even after filtering out words with count < 500, many top-most and bottom-most polarity don't make sense. Identify what kind of words these are and what can be done to filter them out. You can read some tweets in the file to see what's happening.

Textual answer here.

Part-2: Distributional Semantics

Cosine Similarity

Recall that, where i indexes over the context types, cosine similarity is defined as follows. x and y are both vectors of context counts (each for a different word), where x_i is the count of context i.

$$cossim(x, y) = \frac{\sum_i x_i y_i}{\sqrt{\sum_i x_i^2}\sqrt{\sum_i y_i^2}}$$

The nice thing about cosine similarity is that it is normalized: no matter what the input vectors are, the output is between 0 and 1. One way to think of this is that cosine similarity is just, um, the cosine function, which has this property (for non-negative x and y). Another way to think of it is, to work through the situations of maximum and minimum similarity between two context vectors, starting from the definition above.

Note: a good way to understand the cosine similarity function is that the numerator cares about whether the x and y vectors are correlated. If x and y tend to have high values for the same contexts, the numerator tends to be big. The denominator can be thought of as a normalization factor: if all the values of x are really large, for example, dividing by the square root of their sum-of-squares prevents the whole thing from getting arbitrarily large. In fact, dividing by both these things (aka their norms) means the whole thing can’t go higher than 1.

In this problem we'll work with vectors of raw context counts. (As you know, this is not necessarily the best representation.)

Question 5 (5 points)

See the file `nytcounts.university_cat_dog`, which contains context count vectors for three words: “dog”, “cat”, and “university”. These are immediate left and right contexts from a New York Times corpus. You can open the file in a text editor since it’s quite small.

Write a function which takes context count dictionaries of two words and calculates cosine similarity between these two words. The function should return a number beween 0 and 1. Briefly comment on whether the relative simlarities make sense.

In [7]:
```
import distsim;
from distsim import *;
reload(distsim)

word_to_ccdict = distsim.load_contexts("nytcounts.university_cat_dog")
# write code here to show output (i.e. cosine similarity between these words.)
# We encourage you to write other functions in distsim.py itself.
print "Cosine Similarity between dog and cat is:- " + str(cosine_similarity(wo
rd_to_ccdict['cat'], word_to_ccdict['dog']))
print "Cosine Similarity between dog and university is:- " + str(cosine_simila
rity(word_to_ccdict['dog'], word_to_ccdict['university']))
print "Cosine Similarity between cat and university is:- " + str(cosine_simila
rity(word_to_ccdict['university'], word_to_ccdict['cat']))
```

```
file nytcounts.university_cat_dog has contexts for 3 words
Cosine Similarity between dog and cat is:- 0.966891672715
Cosine Similarity between dog and university is:- 0.659230248969
Cosine Similarity between cat and university is:- 0.660442421144
```

Write your response here:

The cobosine similatity here is making a lot of sense. The Cosine Similarity of dog and cat is high given they both are pet. At the same time the cosine similarity of dog and university is comparatively low compared to dog and cat. The same is true for the cat and University is low comparitively of cosine similarity of dog and cat.

Question 6 (20 points)

Explore similarities in `nytcounts.4k`, which contains context counts for about 4000 words in a sample of New York Times articles. The news data was lowercased and URLs were removed. The context counts are for the 2000 most common words in twitter, as well as the most common 2000 words in the New York Times. (But all context counts are from New York Times.) The context counts only contain contexts that appeared for more than one word. The file has three tab-separate fields: the word, its count, and a JSON-encoded dictionary of its context counts. You'll see it's just counts of the immediate left/right neighbors.

Choose **six** words. For each, show the output of 20 nearest words (use cosine similarity as distance metric). Comment on whether the output makes sense. Comment on whether this approach to distributional similarity makes more or less sense for certain terms. Four of your words should be:

- a name (for example: person, organization, or location)
- a common noun
- an adjective
- a verb

You may also want to try exploring further words that are returned from a most-similar list from one of these. You can think of this as traversing the similarity graph among words.

Implementation note: On my laptop it takes several hundred MB of memory to load it into memory from the `load_contexts()` function. If you don't have enough memory available, your computer will get very slow because the OS will start swapping. If you have to use a machine without that much memory available, you can instead implement in a streaming approach by using the `stream_contexts()` generator function to access the data; this lets you iterate through the data from disk, one vector at a time, without putting everything into memory. You can see its use in the loading function. (You could also alternatively use a key-value or other type of database, but that's too much work for this assignment.)

```
In [8]: 'jack' # name
        'jacob' #name
        'paris' # location
        'school' # common noun
        'small' #adjective
        'eat' #verb

Out[8]: 'eat'
```

In [9]:
```
import distsim; reload(distsim)
word_to_ccdict = distsim.load_contexts("nytcounts.4k")
print "\n******* Most 20 most nearest Words to jack*******\n"
distsim.show_nearest(word_to_ccdict, word_to_ccdict['jack'],set(['jack']),dist
sim.cosine_similarity)

print "\n******* Most 20 most nearest Words to jacob*******\n"
distsim.show_nearest(word_to_ccdict, word_to_ccdict['jacob'],set(['jacob']),di
stsim.cosine_similarity)

print "\n******* Most 20 most nearest Words to paris*******\n"
distsim.show_nearest(word_to_ccdict, word_to_ccdict['paris'],set(['paris']),di
stsim.cosine_similarity)

print "\n******* Most 20 most nearest Words to school*******\n"
distsim.show_nearest(word_to_ccdict, word_to_ccdict['school'],set(['school']),
distsim.cosine_similarity)

print "\n******* Most 20 most nearest Words to small*******\n"
distsim.show_nearest(word_to_ccdict, word_to_ccdict['small'],set(['small']),di
stsim.cosine_similarity)

print "\n******* Most 20 most nearest Words to eat*******\n"
distsim.show_nearest(word_to_ccdict, word_to_ccdict['eat'],set(['eat']),distsi
m.cosine_similarity)

###Provide your answer below; perhaps in another cell so you don't have to rel
oad the data each time
```

```
file nytcounts.4k has contexts for 3648 words

******* Most 20 most nearest Words to jack*******

Word = adam and similarity score = 0.879731849466
Word = james and similarity score = 0.859791406534
Word = susan and similarity score = 0.856596258738
Word = daniel and similarity score = 0.847600400729
Word = jonathan and similarity score = 0.847532173876
Word = peter and similarity score = 0.844088466095
Word = eric and similarity score = 0.843254808498
Word = elizabeth and similarity score = 0.843212500388
Word = andrew and similarity score = 0.837502134837
Word = max and similarity score = 0.837313367421
Word = sam and similarity score = 0.83688492819
Word = nancy and similarity score = 0.830201308889
Word = david and similarity score = 0.829875534881
Word = mark and similarity score = 0.825521704605
Word = justin and similarity score = 0.824285104081
Word = thomas and similarity score = 0.815153479018
Word = steven and similarity score = 0.813399333271
Word = henry and similarity score = 0.812411403854
Word = anthony and similarity score = 0.811830749816
Word = chris and similarity score = 0.80986022039

******* Most 20 most nearest Words to jacob*******

Word = max and similarity score = 0.813965842441
Word = elizabeth and similarity score = 0.806544520035
Word = henry and similarity score = 0.804164548253
Word = jack and similarity score = 0.801694099024
Word = honey and similarity score = 0.798835438606
Word = adam and similarity score = 0.790081764384
Word = nike and similarity score = 0.78891744236
Word = daniel and similarity score = 0.78817841634
Word = ohio and similarity score = 0.779273537938
Word = james and similarity score = 0.777941467705
Word = justin and similarity score = 0.77404249882
Word = jonathan and similarity score = 0.763223493263
Word = nyc and similarity score = 0.762117040948
Word = sam and similarity score = 0.760830731607
Word = susan and similarity score = 0.760426508974
Word = chelsea and similarity score = 0.760023820961
Word = thomas and similarity score = 0.756037290295
Word = 2006 and similarity score = 0.753955937751
Word = 34 and similarity score = 0.750406409181
Word = peter and similarity score = 0.749200628067

******* Most 20 most nearest Words to paris*******

Word = london and similarity score = 0.969922701547
Word = 2000 and similarity score = 0.968934319714
Word = washington and similarity score = 0.96828475993
Word = 2002 and similarity score = 0.967796233302
Word = iraq and similarity score = 0.966823178859
Word = 1996 and similarity score = 0.963882680837
Word = baghdad and similarity score = 0.963814951823
Word = 2003 and similarity score = 0.963786580068
Word = 1999 and similarity score = 0.962656787032
Word = 1994 and similarity score = 0.962015475578
Word = 1998 and similarity score = 0.96077176855
Word = 1995 and similarity score = 0.958196421648
Word = 1997 and similarity score = 0.95818436422
Word = europe and similarity score = 0.952949847717
Word = manhattan and similarity score = 0.951685479875
Word = jail and similarity score = 0.94647319085
Word = 2001 and similarity score = 0.945579132203
Word = atlanta and similarity score = 0.942767464066
Word = afghanistan and similarity score = 0.930968363416
Word = september and similarity score = 0.930786955273

******* Most 20 most nearest Words to school*******

Word = schools and similarity score = 0.741096505683
Word = college and similarity score = 0.716161495973
Word = line and similarity score = 0.694353893027
Word = church and similarity score = 0.692936169261
Word = practice and similarity score = 0.692639540488
Word = experience and similarity score = 0.68964261712
Word = location and similarity score = 0.686896635079
Word = scenes and similarity score = 0.684265543442
Word = standards and similarity score = 0.68314025111
Word = movement and similarity score = 0.682270236675
Word = structure and similarity score = 0.681240519859
Word = pain and similarity score = 0.680551450596
Word = club and similarity score = 0.679799771917
Word = star and similarity score = 0.679519599857
Word = trial and similarity score = 0.679030536708
Word = character and similarity score = 0.677712663217
Word = success and similarity score = 0.676811960686
Word = painting and similarity score = 0.676032739005
Word = language and similarity score = 0.673658144796
Word = land and similarity score = 0.67138577287

******* Most 20 most nearest Words to small*******

Word = large and similarity score = 0.973071412011
Word = huge and similarity score = 0.966526892445
Word = rare and similarity score = 0.956221558428
Word = brief and similarity score = 0.954411767655
Word = single and similarity score = 0.951555186585
Word = lovely and similarity score = 0.949123911397
Word = wonderful and similarity score = 0.948595362466
Word = strong and similarity score = 0.945712206547
Word = terrible and similarity score = 0.944205014338
Word = tiny and similarity score = 0.94333832508
Word = special and similarity score = 0.942158289015
Word = giant and similarity score = 0.938978542916
Word = sharp and similarity score = 0.938243642381
Word = little and similarity score = 0.923994722037
Word = fake and similarity score = 0.921158607603
Word = strange and similarity score = 0.920161059203
Word = massive and similarity score = 0.919302821476
Word = broad and similarity score = 0.919022436256
Word = good and similarity score = 0.917984664425
Word = brilliant and similarity score = 0.917552275558

******* Most 20 most nearest Words to eat*******

Word = marry and similarity score = 0.964286212821
Word = shoot and similarity score = 0.963193255852
Word = hide and similarity score = 0.957670012765
Word = stop and similarity score = 0.950446757708
Word = sell and similarity score = 0.94898678915
Word = kill and similarity score = 0.943214395262
Word = buy and similarity score = 0.943124990767
Word = teach and similarity score = 0.942295899732
Word = treat and similarity score = 0.941038126628
Word = win and similarity score = 0.93881787636
Word = grow and similarity score = 0.937942644236
Word = steal and similarity score = 0.935536117618
Word = help and similarity score = 0.935124282998
Word = watch and similarity score = 0.935064832836
Word = write and similarity score = 0.933830419085
Word = pass and similarity score = 0.933271069578
Word = burn and similarity score = 0.932247421537
Word = produce and similarity score = 0.931072999276
Word = draw and similarity score = 0.929059048589
Word = hear and similarity score = 0.926286772812
```

Question 7 (10 points)

In the next several questions, you'll examine similarities in trained word embeddings, instead of raw context counts.

See the file `nyt_word2vec.university_cat_dog`, which contains word embedding vectors pretrained by word2vec [1] for three words: “dog”, “cat”, and “university”, from the same corpus. You can open the file in a text editor since it’s quite small.

Write a function which takes word embedding vectors of two words and calculates cossine similarity between these 2 words. The function should return a number beween -1 and 1. Briefly comment on whether the relative simlarities make sense.

Implementation note: Notice that the inputs of this function are numpy arrays (v1 and v2). If you are not very familiar with the basic operation in numpy, you can find some examples in the basic operation section here: https://docs.scipy.org/doc/numpy-dev/user/quickstart.html (https://docs.scipy.org/doc/numpy-dev/user/quickstart.html)

If you know how to use Matlab but haven't tried numpy before, the following link should be helpful: https://docs.scipy.org/doc/numpy-dev/user/numpy-for-matlab-users.html (https://docs.scipy.org/doc/numpy-dev/user/numpy-for-matlab-users.html)

[1] Mikolov, Tomas, et al. "Distributed representations of words and phrases and their compositionality." NIPS 2013.

In [10]:
```
import distsim; reload(distsim)

word_to_vec_dict = distsim.load_word2vec("nyt_word2vec.university_cat_dog")
print "Cosine Similarity between dog and cat is:- " + str(distsim.cos_sim(word
_to_vec_dict['cat'], word_to_vec_dict['dog']))
print "Cosine Similarity between dog and university is:- " + str(distsim.cos_s
im(word_to_vec_dict['dog'], word_to_vec_dict['university']))
print "Cosine Similarity between cat and university is:- " + str(distsim.cos_s
im(word_to_vec_dict['university'], word_to_vec_dict['cat']))

# write code here to show output (i.e. cosine similarity between these words.)
# We encourage you to write other functions in distsim.py itself.
```

```
Cosine Similarity between dog and cat is:- 0.827517295965
Cosine Similarity between dog and university is:- -0.190753135501
Cosine Similarity between cat and university is:- -0.205394745036
```

Write your response here:

As mentioned above, The cosine similatity here is making a lot of sense. The Cosine Similarity of dog and cat is high given they both are pet. At the same time the cosine similarity of dog and university is comparatively low compared to dog and cat. The same is true for the cat and University is low comparitively of cosine similarity of dog and cat.

Question 8 (20 points)

Repeat the process you did in the question 6, but now use dense vector from word2vec. Comment on whether the outputs makes sense. Compare the outputs of using nearest words on word2vec and the outputs on sparse context vector (so we suggest you to use the same words in question 6). Which method works better on the query words you choose. Please brief explain why one method works better than other in each case.

Not: we used the default parameters of word2vec in gensim (https://radimrehurek.com/gensim/models/word2vec.html) to get word2vec word embeddings.

In [11]:
```
import distsim
word_to_vec_dict = distsim.load_word2vec("nyt_word2vec.4k")
print "\n******* Most 20 most nearest Words to jack*******\n"
distsim.show_nearest(word_to_vec_dict, word_to_vec_dict['jack'],set(['jack']),
distsim.cos_sim)

print "\n******* Most 20 most nearest Words to jacob*******\n"
distsim.show_nearest(word_to_vec_dict, word_to_vec_dict['jacob'],set(['jacob'
]),distsim.cos_sim)

print "\n******* Most 20 most nearest Words to paris*******\n"
distsim.show_nearest(word_to_vec_dict, word_to_vec_dict['paris'],set(['paris'
]),distsim.cos_sim)

print "\n******* Most 20 most nearest Words to school*******\n"
distsim.show_nearest(word_to_vec_dict, word_to_vec_dict['school'],set(['schoo
l']),distsim.cos_sim)

print "\n******* Most 20 most nearest Words to small*******\n"
distsim.show_nearest(word_to_vec_dict, word_to_vec_dict['small'],set(['small'
]),distsim.cos_sim)

print "\n******* Most 20 most nearest Words to eat*******\n"
distsim.show_nearest(word_to_vec_dict, word_to_vec_dict['eat'],set(['eat']),di
stsim.cos_sim)

###Provide your answer below; perhaps in another cell so you don't have to rel
oad the data each time
###Provide your answer below
```

```
******* Most 20 most nearest Words to jack*******

Word = sam and similarity score = 0.804650868501
Word = jim and similarity score = 0.784263389501
Word = adam and similarity score = 0.777474342932
Word = ed and similarity score = 0.775161593077
Word = chris and similarity score = 0.770623148763
Word = anthony and similarity score = 0.759454281466
Word = bruce and similarity score = 0.748197814965
Word = brian and similarity score = 0.745924331721
Word = steve and similarity score = 0.744650198971
Word = ray and similarity score = 0.744424710639
Word = bob and similarity score = 0.740298867859
Word = jonathan and similarity score = 0.739568545763
Word = matt and similarity score = 0.738397895777
Word = larry and similarity score = 0.729699086832
Word = daniel and similarity score = 0.729641022873
Word = josh and similarity score = 0.72950135195
Word = jeff and similarity score = 0.728281357786
Word = alan and similarity score = 0.727446889115
Word = eric and similarity score = 0.722989518866
Word = gary and similarity score = 0.722301102389

******* Most 20 most nearest Words to jacob*******

Word = elizabeth and similarity score = 0.846903711683
Word = max and similarity score = 0.836984459923
Word = clifford and similarity score = 0.828206166377
Word = leo and similarity score = 0.825521942265
Word = k. and similarity score = 0.8141050643
Word = susan and similarity score = 0.804036295633
Word = andrew and similarity score = 0.801292418996
Word = jonathan and similarity score = 0.781430133583
Word = t. and similarity score = 0.780036398061
Word = henry and similarity score = 0.777814773613
Word = adam and similarity score = 0.771475584181
Word = b. and similarity score = 0.768043166594
Word = lawrence and similarity score = 0.767618660228
Word = anthony and similarity score = 0.764368520289
Word = justin and similarity score = 0.763949007361
Word = barbara and similarity score = 0.762113229322
Word = jay and similarity score = 0.757096729658
Word = robin and similarity score = 0.75676982718
Word = edward and similarity score = 0.755632189535
Word = daniel and similarity score = 0.75370633517

******* Most 20 most nearest Words to paris*******

Word = london and similarity score = 0.742107827129
Word = spain and similarity score = 0.634463364795
Word = australia and similarity score = 0.623465314272
Word = italy and similarity score = 0.595381536063
Word = france and similarity score = 0.58273759425
Word = la and similarity score = 0.546190042754
Word = germany and similarity score = 0.535940620503
Word = el and similarity score = 0.531092441066
Word = argentina and similarity score = 0.526052579318
Word = madrid and similarity score = 0.522976187947
Word = chelsea and similarity score = 0.522678936055
Word = hotel and similarity score = 0.517737491196
Word = chicago and similarity score = 0.504221789448
Word = restaurant and similarity score = 0.497719991719
Word = japan and similarity score = 0.485197298741
Word = royal and similarity score = 0.469962805048
Word = 1960 and similarity score = 0.466812750658
Word = del and similarity score = 0.465245854511
Word = 1996 and similarity score = 0.465106626993
Word = de and similarity score = 0.46145017425

******* Most 20 most nearest Words to school*******

Word = schools and similarity score = 0.75222831832
Word = college and similarity score = 0.749073141248
Word = class and similarity score = 0.62663670792
Word = student and similarity score = 0.580021551051
Word = classes and similarity score = 0.55825512395
Word = columbia and similarity score = 0.546274281597
Word = teacher and similarity score = 0.536806366718
Word = academy and similarity score = 0.534093683027
Word = university and similarity score = 0.515488570733
Word = students and similarity score = 0.511512873819
Word = education and similarity score = 0.511472003069
Word = harvard and similarity score = 0.5090504665
Word = tech and similarity score = 0.50430522084
Word = math and similarity score = 0.503104202652
Word = teaching and similarity score = 0.493482811826
Word = teachers and similarity score = 0.486860886982
Word = princeton and similarity score = 0.485218743452
Word = yale and similarity score = 0.469128724487
Word = gym and similarity score = 0.467681661412
Word = degree and similarity score = 0.452242066307

******* Most 20 most nearest Words to small*******

Word = large and similarity score = 0.872116601448
Word = tiny and similarity score = 0.740835894895
Word = vast and similarity score = 0.664048960627
Word = huge and similarity score = 0.641445380654
Word = smaller and similarity score = 0.626264603893
Word = big and similarity score = 0.58646670179
Word = larger and similarity score = 0.583728744062
Word = separate and similarity score = 0.554064402132
Word = massive and similarity score = 0.550263049553
Word = wide and similarity score = 0.514304266979
Word = private and similarity score = 0.508697739441
Word = broad and similarity score = 0.506540355212
Word = steel and similarity score = 0.501893723321
Word = traditional and similarity score = 0.496682179926
Word = mostly and similarity score = 0.495191384289
Word = variety and similarity score = 0.475786275018
Word = limited and similarity score = 0.47191176752
Word = heavy and similarity score = 0.471686086364
Word = rare and similarity score = 0.468135909424
Word = significant and similarity score = 0.467857634805
```

```
******* Most 20 most nearest Words to eat*******

Word = drink and similarity score = 0.765148798351
Word = enjoy and similarity score = 0.711493132019
Word = sleep and similarity score = 0.706877663891
Word = feed and similarity score = 0.685293152453
Word = breathe and similarity score = 0.673759067476
Word = wear and similarity score = 0.670034287822
Word = forget and similarity score = 0.658329445839
Word = ate and similarity score = 0.65470653416
Word = burn and similarity score = 0.634897570213
Word = get and similarity score = 0.614431792198
Word = eating and similarity score = 0.613962036626
Word = treat and similarity score = 0.60318544897
Word = smell and similarity score = 0.603163870658
Word = buy and similarity score = 0.602649048162
Word = listen and similarity score = 0.595459586665
Word = sit and similarity score = 0.594974533206
Word = see and similarity score = 0.587573346075
Word = cook and similarity score = 0.585612527119
Word = stick and similarity score = 0.581404640223
Word = hang and similarity score = 0.580721791486
```

Question 9 (15 points)

An interesting thing to try with word embeddings is analogical reasoning tasks. In the following example, it's intended to solve the analogy question "king is to man as what is to woman?", or in SAT-style notation,

king : man :: __ : woman

Some research has proposed to use additive operations on word embeddings to solve the analogy: take the vector $(v_{king} - v_{man} + v_{woman})$ and find the most-similar word to it. One way to explain this idea: if you take "king", get rid of its attributes/contexts it shares with "man", and add in the attributes/contexts of "woman", hopefully you'll get to a point in the space that has king-like attributes but the "man" ones replaced with "woman" ones.

Show the output for 20 closest words you get by trying to solve that analogy with this method. Did it work?

Please come up with another analogical reasoning task (another triple of words), and output the answer using the same method. Comment on whether the output makes sense. If the output makes sense, explain why we can capture such relation between words using an unsupervised algorithm. Where does the information come from? On the other hand, if the output does not make sense, propose an explanation why the algorithm fails on this case.

Note that the word2vec is trained in an unsupervised manner just with distributional statistics; it is interesting that it can apparently do any reasoning at all. For a critical view, see Linzen 2016 (http://www.aclweb.org/anthology/W/W16/W16-2503.pdf).

In [12]:
```
# Write code to show output here.
import distsim
king = word_to_vec_dict['king']
man = word_to_vec_dict['man']
woman = word_to_vec_dict['woman']
distsim.show_nearest(word_to_vec_dict,
                     king-man+woman,
                     set(['king','man','woman']),
                     distsim.cos_sim)
```

```
Word = queen and similarity score = 0.725028631986
Word = princess and similarity score = 0.577900103401
Word = prince and similarity score = 0.566962392417
Word = lord and similarity score = 0.530919391111
Word = royal and similarity score = 0.520203296864
Word = mary and similarity score = 0.497698146284
Word = mama and similarity score = 0.495469636832
Word = daughter and similarity score = 0.493757946566
Word = singer and similarity score = 0.489838082014
Word = kim and similarity score = 0.488354695243
Word = elizabeth and similarity score = 0.482484843405
Word = girl and similarity score = 0.477338294808
Word = grandma and similarity score = 0.476990726681
Word = sister and similarity score = 0.470304371825
Word = mother and similarity score = 0.469422028833
Word = clark and similarity score = 0.46824004741
Word = wedding and similarity score = 0.46233629356
Word = husband and similarity score = 0.456851188179
Word = boyfriend and similarity score = 0.447550574504
Word = jesus and similarity score = 0.438572115806
```

In [13]:
```
japan = word_to_vec_dict['japan']
brazil = word_to_vec_dict['brazil']
france = word_to_vec_dict['france']
distsim.show_nearest(word_to_vec_dict,
                     japan-brazil+france,
                     set(['japan','brazil','france']),
                     distsim.cos_sim)
```

```
Word = germany and similarity score = 0.852228240177
Word = britain and similarity score = 0.796347823164
Word = europe and similarity score = 0.780507381157
Word = italy and similarity score = 0.780123880453
Word = spain and similarity score = 0.744563107146
Word = india and similarity score = 0.741158443781
Word = russia and similarity score = 0.71760613491
Word = australia and similarity score = 0.689681638742
Word = argentina and similarity score = 0.680727627136
Word = china and similarity score = 0.680026975466
Word = canada and similarity score = 0.637877286391
Word = america and similarity score = 0.61665435954
Word = european and similarity score = 0.605925266688
Word = africa and similarity score = 0.605411548357
Word = ukraine and similarity score = 0.59694393205
Word = afghanistan and similarity score = 0.586397734305
Word = french and similarity score = 0.573833854437
Word = paris and similarity score = 0.570905710291
Word = iran and similarity score = 0.555832473756
Word = german and similarity score = 0.542073014482
```

Textual answer here.

Yes the output is making sense in the above analogy, the result should be list of contries, which the word2vec model is return flawlessly. The model learns to map each discrete word id (0 through the number of words in the vocabulary) into a low-dimensional continuous vector-space from their distributional properties observed in some raw text corpus. Geometrically, one may interpret these vectors as tracing out points on the outside surface of a manifold in the "embedded space".

In [ ]:
```
```