Link Prediction in the Greek Web

In this notebook we are dealing with the problem of link prediction in graphs.Specifically we have a subset of the Greek web and we are trying to predict missing edges. The approach we are going to follow is to deal with the problem as a binary classification task, where each edge is a candidate new edge for the graph. In the specific dataset there are 2041 nodes and 2683 edges and we are trying to predict the 453 edges that have been manually removed from the graph. We also have a dataset of different texts available from the hosts.

First we extract the raw text from the hosts

In [108]:
```
import os
import zipfile
import nltk
import pickle

if os.path.isfile('cache/processed_text.pickle'):
    with open('cache/processed_text.pickle', 'rb') as pfile:
        text_data = pickle.load(pfile)
        print "loaded from pickle"
else:
    filenames = os.listdir('dataset/hosts')
    raw_text = {}
    for zipfilename in filenames:
        with zipfile.ZipFile('dataset/hosts/'+zipfilename) as z:
            text = ""
            for filename in z.namelist():
                if not os.path.isdir(filename):
                    with z.open(filename) as f:
                        for line in f:
                            text += line.decode("utf-8").upper()
                            text += " "
            raw_text[zipfilename[:-4]] = text
    text_data = process_text(raw_text)
    with open('cache/processed_text.pickle', 'wb') as pfile:
        pickle.dump(text_data, pfile)
```

```
loaded from pickle
```

The function process_word is used for stemming as well as to remove tones from the greek text.

In [109]:
```
import stemmer as st
def process_word(word):
    """detone and stem word
    """
    new_word = []
    tones = { u'Ά' : u'Α' , u'Έ': u'Ε', u'Ί': u'Ι', u'Ϊ': u'Ι',
             u'Ύ': u'Υ', u'Ϋ': u'Υ', u'Ό' : u'Ο', u'Ή': u'Η', u'Ώ': u'Ω'}
    for letter in word:
        try:
            new_word.append(tones[letter])
        except KeyError:
            new_word.append(letter)
    detoned = ''.join(l for l in new_word)
    return st.stem(detoned)
```

The function process_text is being used to remove stopwords, punctuation, numbers as well as use the above function for detoning and stemming.

In [110]:
```
import string
def process_text(data):
    with open('dataset/greekstopwords.txt', 'r') as fp:
        stopwords = []
        for line in fp:
            stopwords.append(line.strip().decode('utf-8').upper())
    for domain in data.keys():
        text = data[domain]
        # remove punctuation
        punctuation = set(string.punctuation)
        doc = ''.join([w for w in text if w not in punctuation])
        # remove stopwords
        doc = [w for w in doc.split() if w not in stopwords]
        doc = [w for w in doc if not re.match(r"$\d+\W+|\b\d+\b|\W+\d+$", w)]
        doc = ' '.join(process_word(w) for w in doc)
        data[domain] = doc
    return data
```

In [111]:
```
# assign each domain to the number.
domain_number = {}
for i, domain in enumerate(text_data.keys()):
    domain_number[domain] = i
```

Afterwards we extract tfidf features from the websites. We choose 500 words as features and we compute the pairwise cosine similarity for all the websites. Below we also print those features.

In [112]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity

vec = TfidfVectorizer(max_df=0.90, min_df=5, max_features=500,lowercase=False ,
                      analyzer = 'word')
x = vec.fit_transform(text_data.values())
for word in vec.get_feature_names():
    print word,
cosine = cosine_similarity(x)
```

```
ABOUT ALL AND ARE AT ATHENS ΑΔΕΙ ΑΛΛ ΑΝΘΡΩΠ ΑΡΘΡ ATOM BLOG BY COMMENT COMMENTS CONTACT COOKIES COPYRIGHT DE DESIGN
EMAIL FACEBOOK FM FOR FROM GOOGLE GREECE GREEK HOME HOT IN IS IT JULY LED LIFESTYLE LIKE LIVE MAY MEDIA MORE NEW NE
WS NEWSLETTER NO OCTOBER OF ON ONLINE OR OUT POST POSTED POWERED RADIO READ RESERVED RIGHTS RSS SEARCH SEPTEMBER SH
ARE SITE THE THIS TO TOP TWEET TWITTER US VIDEO VIEW WEB WITH YOU YOUR ΑΓ ΑΓΟΡ ΑΓΟΡΑ ΑΓΩΝ ΑΘΗΝ ΑΘΛΗΤ ΑΘΛΗΤΙΣΜ ΑΚΟΛΟ
ΥΘ ΑΚΟΜ ΑΛΛ ΑΛΛΑ ΑΝΑΠΤΥΞ ΑΝΑΖΗΤΗΣ ΑΝΑΚΟΙΝΩΣ ΑΝΘΡΩΠ ΑΠ ΑΠΑΝΤΗΣ ΑΠΟ ΑΠΟΣΤΟΛ ΑΠΟΤΕΛ ΑΠΟΤΕΛΕΣΜ ΑΠΟΦΑΣ ΑΠΟΦΑΣ ΑΠΟΨ ΑΠΡΙΛ
ΑΠΡΙΛΙ ΑΡΘΡ ΑΡΙΘΜ ΑΡΧ ΑΡΧΕΙ ΑΣΦΑΛΕΙ ΑΤΤ ΑΥΓ ΑΥΓΟΥΣΤ ΑΥΤ ΑΥΤΑ ΑΥΤΟΚΙΝΗΤ ΑΦ ΒΑΣ ΒΑΘΜΟΛΟΓ ΒΙΒΛ ΒΙΒΛΙ ΒΙΝΤΕ ΒΟΛ ΒΡ ΒΡΙΣ
Κ ΓΕΝ ΓΕΡΜΑΝ ΓΕΩΡΓΙ ΓΙΑΝΝ ΓΙΑΤ ΓΙΝ ΓΙΩΡΓ ΓΝΩΣΤ ΓΟΝ ΓΡΑΦ ΓΡΑΦΕΙ ΓΡΗΓΟΡ ΓΥΝΑΙΚ ΔΕ ΔΕΔΟΜΕΝ ΔΕΚ ΔΕΚΕΜΒΡ ΔΗΛΩΣ ΔΗΜ ΔΗΜΗΤ
Ρ ΔΗΜΙΟΥΡΓ ΔΗΜΟΣ ΔΗΜΟΣΙ ΔΗΜΟΣΙΕΥΘ ΔΗΜΟΣΙΕΥΣ ΔΗΜΟΤ ΔΗΜΟΦΙΛ ΔΙΑΒΑΣ ΔΙΑΡΚΕΙ ΔΙΑΦΟΡ ΔΙΑΒΑΣ ΔΙΑΒΑΣΤ ΔΙΑΓΩΝΙΣΜ ΔΙΑΤΡΟΦ ΔΙ
ΑΦΗΜΙΣ ΔΙΑΧΕΙΡΙΣ ΔΙΕΘΝ ΔΙΕΥΘΥΝΣ ΔΙΚΑΙ ΔΙΚΑΙΩΜ ΔΙΚΤΥ ΔΙΝ ΔΙΟΙΚΗΣ ΔΡΑΣ ΔΡΑΣΤΗΡΙΟΤΗΤ ΔΡΟΜ ΔΥΝΑΜ ΔΥΝΑΤΟΤΗΤ ΔΥΟ ΔΥΣΚΟΛ Δ
ΩΡΕΑΝ ΕΒΔΟΜΑΔ ΕΓΓΡΑΦ ΕΓΙΝ ΕΔΩ ΕΘΝ ΕΙΔ ΕΙΔΗΣ ΕΙΚΟΝ ΕΙΜΑΣΤ ΕΙΝΑ ΕΙΠ ΕΙΧ ΕΚ ΕΚΔΗΛΩΣ ΕΚΔΟΣ ΕΚΕΙΝ ΕΚΘΕΣ ΕΚΛΟΓ ΕΚΠΑΙΔΕΥΣ
ΕΚΠΑΙΔΕΥΤ ΕΚΤ ΕΛΕΓΧ ΕΛΕΥΘΕΡ ΕΛΛΑΔ ΕΛΛΑΔ ΕΛΛΗΝ ΕΛΛΗΝΙΚ ΕΝ ΕΝΑ ΕΝΕΡΓΕΙ ΕΝΗΜΕΡΩΣ ΕΝΟΤΗΤ ΕΝΩ ΕΝΩΣ ΕΞΩΤΕΡ ΕΠΙ ΕΠΙΚΑΙΡΟΤΗ
Τ ΕΠΙΚΟΙΝΩΝ ΕΠΙΛΟΓ ΕΠΙΣ ΕΠΙΣΚΕΨ ΕΠΙΤΡΕΠ ΕΠΙΤΡΟΠ ΕΠΙΧΕΙΡΗΣ ΕΠΟΜΕΝ ΕΠΟΧ ΕΡΓ ΕΡΓΑΣ ΕΡΓΑΣΙ ΕΡΕΥΝ ΕΡΧ ΕΡΩΤΗΣ ΕΤ ΕΤΑΙΡ ΕΤ
ΑΙΡΕΙ ΕΤΣ ΕΥΚΟΛ ΕΥΡ ΕΥΡΩΠ ΕΥΡΩΠΑΙΚ ΕΦΑΡΜΟΓ ΕΦΗΜΕΡΙΔ ΕΧ ΕΩΣ ΖΗΤ ΖΩ ΖΩΗ ΗΛΕΚΤΡΟΝ ΗΜΕΡ ΗΠΑ ΗΤ ΘΕΑΤΡ ΘΕΛ ΘΕΜ ΘΕΣ ΘΕΣΣΑΛ
ΟΝ ΘΕΣΣΑΛΟΝΙΚ ΘΕΩΡ ΙΑΤΡ ΙΔ ΙΔΙ ΙΔΙΑΙΤΕΡ ΙΟΥΛ ΙΟΥΛΙ ΙΟΥΝ ΙΟΥΝΙ ΙΣΤΟΡ ΙΣΤΟΣΕΛΙΔ ΚΑΘ ΚΑΝ ΚΑΠΟΙ ΚΑΤ ΚΑΘ ΚΑΙΡ ΚΑΛ ΚΑΛΑΘ
ΚΑΝ ΚΑΤΑ ΚΑΤΑΣΚΕΥ ΚΑΤΗΓΟΡ ΚΑΤΗΓΟΡΙ ΚᾺΙ ΚΕΝΤΡ ΚΙΝ ΚΛΙΚ ΚΟΙΝ ΚΟΙΝΟΤΗΤ ΚΟΙΝΩΝ ΚΟΣΜ ΚΡΗΤ ΚΡΙΣ ΚΥΒΕΡΝΗΣ ΚΥΠΡ ΚΥΡΙ ΚΥΡΙΑΚ
ΚΩΔ ΚΩΝΣΤΑΝΤΙΝ ΛΕ ΛΕΙΤΟΥΡΓ ΛΙΓ ΛΙΣΤ ΛΟΓ ΛΟΓΑΡΙΑΣΜ ΛΥΣ ΜΑΘ ΜΑΙ ΜΑΡΤΙ ΜΑΐ́ ΜΑΖ ΜΑΘΗΤ ΜΑΡ ΜΑΡΤ ΜΕΓΑΛ ΜΕΓΑΛ ΜΕΛ ΜΕΡ ΜΕΣ
ΜΕΤΑ ΜΕΤΑΞ ΜΕΤΡ ΜΕΧΡ ΜΗΝ ΜΗΝΥΜ ΜΙΑ ΜΙΚΡ ΜΜ ΜΟΙΡΑΣΤ ΜΟΝ ΜΟΝΑΔ ΜΟΥΣ ΜΟΥΣΕΙ ΜΠΟΡ ΝΕ ΝΕΑ ΝΕΟ ΝΙΚΟΛΑ ΝΟΕΜΒΡ ΝΟΜ ΞΕΚΙΝ ΞΕ
ΝΟΔΟΧΕΙ ΞΕΡ ΟΔΗΓ ΟΙΚΟΓΕΝΕΙ ΟΙΚΟΝΟΜ ΟΙΚΟΝΟΜΙΚ ΟΚΤ ΟΚΤΩΒΡ ΟΚΤΩΒΡΙ ΟΛ ΟΛΑ ΟΛΗ ΟΛΟΚΛΗΡ ΟΛΥΜΠΙΑΚ ΟΜ ΟΜΑΔ ΟΜΙΛ ΟΝΟΜ ΟΠ ΟΠ
ΟΙ ΟΡ ΟΡΓΑΝΙΣΜ ΟΣ ΟΣΟ ΟΤΙ ΟΥΤ ΟΧΙ ΠΑΝ ΠΑΝΤ ΠΑΡ ΠΑΓΚΟΣΜ ΠΑΓΚΟΣΜΙ ΠΑΙΔ ΠΑΙΔΙΑ ΠΑΙΧΝΙΔ ΠΑΡΑΓΩΓ ΠΑΡΑΣΚΕΥ ΠΕΜΠΤ ΠΕΡ ΠΕΡΙ
ΕΧΟΜΕΝ ΠΕΡΙΟΔ ΠΕΡΙΟΧ ΠΕΡΙΠΤΩΣ ΠΕΡΙΣΣ ΠΕΡΙΦΕΡΕΙ ΠΗΓ ΠΛΕ ΠΛΗΡΟΦΟΡΙ ΠΜ ΠΟΔΟΣΦΑΙΡ ΠΟΙΟΤΗΤ ΠΟΛ ΠΟΛΙΤ ΠΟΛΙΤΙΚ ΠΟΛΙΤΙΣΜ ΠΟ
ΛΛ ΠΟΛΛΑ ΠΟΣ ΠΟΤ ΠΡΑΓΜΑΤΟΠΟΙ ΠΡΕΠ ΠΡΟΒΛΗΜ ΠΡΟΒΟΛ ΠΡΟΓΡΑΜΜ ΠΡΟΓΡΑΜΜ ΠΡΟΕΔΡ ΠΡΟΗΓΟΥΜΕΝ ΠΡΟΙΟΝΤ ΠΡΟΣΚΛΗΣ ΠΡΟΣΦΑΤ ΠΡΟΣΦ
ΕΡ ΠΡΟΣΦΟΡ ΠΡΟΣΩΠ ΠΡΟΤΑΣ ΠΡΩΤ ΠΩΣ ΣΑΒΒΑΤ ΣΕΛΙΔ ΣΕΠ ΣΕΠΤΕΜΒΡ ΣΕΠΤΕΜΒΡΙ ΣΗΜΕΙ ΣΗΜΕΡ ΣΠΙΤ ΣΤΑΘΜ ΣΤΗΛ ΣΤΙΓΜ ΣΤΟΙΧΕΙ ΣΤΟ
Χ ΣΥΓΚΕΚΡΙΜΕΝ ΣΥΛΛΟΓ ΣΥΜΒΟΥΛ ΣΥΜΒΟΥΛΙ ΣΥΜΜΕΤΟΧ ΣΥΜΦΩΝ ΣΥΝΑΝΤΗΣ ΣΥΝΔΕΣ ΣΥΝΔΕΣΜ ΣΥΝΕΔΡΙ ΣΥΝΕΔΡΙΑΣ ΣΥΝΕΝΤΕΥΞ ΣΥΝΕΧΕΙ Σ
ΥΝΤΑΓ ΣΥΡΙΖ ΣΥΣΤΗΜ ΣΧΕΔ ΣΧΕΣ ΣΧΕΤΙΚΑ ΣΧΟΛ ΣΧΟΛΕΙ ΣΧΟΛΙ ΣΧΟΛΙΑΣΜ ΣΩΜ ΤΑΚΤ ΤΑΞΙΔ ΤΕΛ ΤΕΛΕΥΤΑΙ ΤΕΤΑΡΤ ΤΕΥΧ ΤΕΧΝ ΤΕΧΝΟΛ
ΟΓ ΤΗΛ ΤΗΛΕΦΩΝ ΤΙΜ ΤΙΤΛ ΤΜΗΜ ΤΟ ΤΟΠ ΤΟΣ ΤΟΤ ΤΟΥΡΙΣΜ ΤΟΥΡΚ ΤΟῦ ΤΡ ΤΡΑΠΕΖ ΤΡΑΓΟΥΔ ΤΡΙΤ ΤΡΟΠ ΤΣΙΠΡ ΤΥΠ ΤΩΡ ΥΓΕΙ ΥΛ ΥΠΑ
ΡΧ ΥΠΗΡΕΣ ΥΠΗΡΕΣΙ ΥΠΟΥΡΓ ΥΨΗΛ ΦΕΣΤΙΒΑΛ ΦΙΛ ΦΟΡ ΦΟΡΑ ΦΥΣΙΚ ΦΩΤ ΦΩΤΟΓΡΑΦ ΦΩΤΟΓΡΑΦΙ ΧΑΡΤ ΧΡΕ ΧΡΗΣ ΧΡΗΣΙΜ ΧΡΗΣΙΜΟΠΟΙ ΧΡ
ΗΣΤ ΧΡΟΝ ΧΡΥΣ ΧΩΡ ΧΩΡΙΣ ΩΡ ΩΡΑ
```

In [113]:
```
# testing the text similarity of two websites
cosine[domain_number['news247.gr'], domain_number['newsit.gr']]
```

Out[113]: 0.4863206743113806

In [114]:
```
from __future__ import division
import networkx as nx
import pprint
import random
import numpy as np
from scipy import sparse
from sklearn.model_selection import train_test_split
```

In [115]:
```
def text_similarity(src, dst):
    return cosine[domain_number[src], domain_number[dst]]
```

Topic Extraction and Document Clustering

In [116]:
```
# from time import time
# from sklearn.decomposition import NMF, LatentDirichletAllocation
# tfidf_vec = TfidfVectorizer(max_df=0.90, min_df=5, max_features=500,lowercase=False ,
#                            analyzer = 'word')
# tfidf = tfidf_vec.fit_transform(text_data.values())
# tf_vec = TfidfVectorizer(max_df=0.90, min_df=5, max_features=500,lowercase=False ,
#                            analyzer = 'word')
# tf = tf_vec.fit_transform(text_data.values())
```

In [117]:
```
# t0 = time()
# nmf = NMF(n_components=50, random_state=1,
#           alpha=.1, l1_ratio=.5).fit(tfidf)
# print("done in %0.3fs." % (time() - t0))
```

In [118]:
```
# def print_top_words(model, feature_names, n_top_words):
#     for topic_idx, topic in enumerate(model.components_):
#         print("Topic #%d:" % topic_idx)
#         print(" ".join([feature_names[i]
#                         for i in topic.argsort()[:-n_top_words - 1:-1]]))
#     print()
```

In [119]:
```
# print("\nTopics in NMF model:")
# tfidf_feature_names = tfidf_vec.get_feature_names()
# print_top_words(nmf, tfidf_feature_names, 10)
```

In [120]:
```
# lda = LatentDirichletAllocation(n_topics=605, max_iter=5,
#                                 learning_method='online',
#                                 learning_offset=50.,
#                                 random_state=0)
# t0 = time()
# lda.fit(tf)
# print("done in %0.3fs." % (time() - t0))

# print("\nTopics in LDA model:")
# tf_feature_names = tf_vec.get_feature_names()
# # print_top_words(lda, tf_feature_names, 10)
```

In [121]:
```
# dist = 1 -cosine
# from sklearn.cluster import KMeans

# num_clusters = 605
# km = KMeans(n_clusters=num_clusters)
# km.fit(tfidf)
# cluster_assignment = km.labels_.tolist()
```

In [122]:
```
# clusters = {i : [] for i in cluster_assignment}
# for node, cluster in enumerate(cluster_assignment):
#     clusters[cluster].append(G.nodes()[node])
```

In [123]:
```
# import pprint
# pprint.pprint(clusters,width=80)
```

Graph

At first we create the directed graph from the 'edgelist.txt'.

In [124]:
```
import networkx as nx
import pprint

G = nx.read_edgelist('dataset/edgelist.txt', delimiter='\t', create_using=nx.DiGraph())
print len(G.nodes()), " number of nodes"
print len(G.edges()), " number of edges"
```

```
2041  number of nodes
2683  number of edges
```

In [125]:
```
# random.seed(1)
# edges = []
# with open('dataset/edgelist.txt', 'r')as fp:
#     for line in fp:
#         edges.append((line.split()[0], line.split()[1]))
# print len(edges)
# nodes = set([node for _tuple in edges for node in _tuple])
# print len(nodes)
```

Train and Test Data

Training Set: For the training set we use 20760 edges that we are sure they do not exist (given in the 'non_existing_edges.txt' as the 1nd category and we use the 2683 edges from the graph which are the 2nd category for our classification problem.

Test Set: We use the 4160957 edges that do not exist in the graph. From those we remove the 20760 edges that we know that don't exist to end up with 4140197 candidate edges.

For the edges available in the test set we will use a classifier to predict the class for each edge(0 or 1). We are interested in the probability that an edge exists. Afterwards we sort these probabilities and obtain the top 453 edges that are used for the submission.
An alternate approach would be to use a number of random edges for the training set as nonexistent edges assuming that since the selective rate of such a set would be small that it would not affect as much the accuracy score.

In [126]:
```
#load the edges that do not exist
non_existent_edges = {}
with open('dataset/not_existing_edges.txt', 'r')as fp:
    for line in fp:
        non_existent_edges[((line.split()[0], line.split()[1]))] = 0
len(non_existent_edges)
```

Out[126]: 20760

In [127]:
```
non_edges = [edge for edge in list(nx.non_edges(G)) if not non_existent_edges.has_key(edge)]
#4160957
len(non_edges)
```

Out[127]: 4140197

In [128]:
```
# #new test with random edges
# #non_edges = [edge for edge in list(nx.non_edges(G))]
# non_existent_edges = {}
# random_numbers = [random.randint(0,len(non_edges)) for i in range(150000)]
# for i in random_numbers:
#     non_existent_edges[((non_edges[i][0], non_edges[i][1]))] = 0
```

Extracting Features

We use networkx to help us extract useful features for each node of the graph

In [129]:
```
pagerank = nx.pagerank(G)
betweeness = nx.betweenness_centrality(G)
closeness = nx.closeness_centrality(G)
eigenvector = nx.eigenvector_centrality(G)
degree = nx.degree_centrality(G)
in_degree = G.in_degree()
out_degree = G.out_degree()
katz = nx.katz_centrality(G)
core_number = nx.core_number(G)
triangles = nx.triangles(G.to_undirected())
```

In [130]:
```
import graphsim as gs
simrank = gs.simrank(G)
```

```
Converge after 20 iterations (eps=0.000100).
```

In [131]:
```
import math
def adamic_adar(src, dst):
    score = 0
    common = list(set(G.neighbors(src)).intersection(G.neighbors(dst)))
    return sum(1 / math.log(G.degree(w))
               for w in common)
```

In [132]:
```
adamic_adar(u'news247.gr', u'contra.gr')
```

Out[132]:
```
3.0020540579927957
```

In [133]:
```
un_g = G.to_undirected()
```

Community Detection

We perform community detection on the graph using the louvain method. We obtain 605 number of communities. By examining them that some of them are really similar. For example they may belong to the same company, or they may have semantic similarity. We use these communities to extract two features:
Partition_common: declares the number of nodes from one's nodes neighborhood that exist in the same community as the other.

In [134]:
```
import igraph as ig
import louvain_igraph as louvain
G_new = ig.Graph()
G_new.add_vertices(G.nodes())
G_new.add_edges(G.edges())
```

In [135]:
```
opt = louvain.Optimiser()
partition = opt.find_partition(graph=G_new,partition_class=louvain.SignificanceVertexPartition)
```

In [136]:
```
partition = list(partition)
partition_names = []
for com in partition:
    new_com = []
    for node_id in com:
        new_com.append(G_new.vs[node_id]['name'])
    partition_names.append(new_com)
# extract names
partitions = { node : [] for node in G.nodes()}
for com in partition_names:
    for node in com:
        partitions[node].extend(com)
print len(partition)
```

```
620
```

In [137]:
```
"""for com in partition_names:
    if com:
        print com"""
```

Out[137]:
```
'for com in partition_names:\n    if com:\n        print com'
```

In [138]:

```
def partition_common(src, dst):
    counter = 0
    for neigh in G.neighbors(src):
        if neigh in partitions[dst]:
            counter+=1
    for neigh in G.neighbors(dst):
        if neigh in partitions[src]:
            counter+=1
    return counter

def partition_check(src, dst):
    if src in partitions[dst]:
        return 1
    else:
        return 0

partition_common('news247.gr', 'ladylike.gr')
```

Out[138]: 17

In [139]:

```
def second_neighbors(src, dst):
    """
    returns the number of common second level neighbors between two nodes"""
    level1_src = G.neighbors(src)
    level1_dst = G.neighbors(dst)
    score = 0
    level2_src = []
    level2_dst = []
    for w in level1_src:
        level2_src.extend(G.neighbors(w))
    for w in level1_dst:
        level2_dst.extend(G.neighbors(w))
    common = list(set(level2_src).intersection(level2_dst))
    return len(common)
```

In [140]:

```
second_neighbors('news247.gr', 'contra.gr')
```

Out[140]: 15

In [141]:

```
from scipy import stats
print "pagerank", stats.describe(pagerank.values())
print "closeness", stats.describe(closeness.values())
print "betweeness", stats.describe(betweeness.values())
print "eigenvector", stats.describe(eigenvector.values())
print "text similarity", stats.describe(cosine)
```

```
pagerank DescribeResult(nobs=2041, minmax=(0.00029761511829001677, 0.019530823306958298), mean=0.000489955903968642
49, variance=5.1484022288357726e-07, skewness=16.593681826353492, kurtosis=379.97825697924446)
closeness DescribeResult(nobs=2041, minmax=(0.0, 0.043706197338373103), mean=0.0011057427491154147, variance=1.0548
295767353478e-05, skewness=6.840389761601526, kurtosis=62.844402531343476)
betweeness DescribeResult(nobs=2041, minmax=(0.0, 0.0013256810816528672), mean=5.1837164079498881e-06, variance=3.5
261533531813181e-09, skewness=16.489608369782463, kurtosis=294.5096041003185)
eigenvector DescribeResult(nobs=2041, minmax=(0.0, 0.37735422231142191), mean=0.0017948109226848491, variance=0.000
48697315309188523, skewness=13.386178390197925, kurtosis=183.2693369063938)
text similarity DescribeResult(nobs=2041, minmax=(array([ 0.,  0.,  0., ...,  0.,  0.,  0.]), array([ 1.,  1.,  1.,
...,  1.,  1.,  1.])), mean=array([ 0.14081705,  0.11499577,  0.19841769, ...,  0.18956786,
        0.16056721,  0.03456392]), variance=array([ 0.00814462,  0.0042941 ,  0.01530137, ...,  0.01352619,
        0.01198027,  0.00277865]), skewness=array([ 1.13119908,  1.64330689,  0.9237617 , ...,  0.47913089,
        0.79946699,  6.66819173]), kurtosis=array([  4.41391101,  15.89084406,   1.30397545, ...,   0.49640679,
         1.50261023,  81.78381561]))
```

In [142]:

```
nx.adamic_adar_index(un_g, [('news247.gr', 'contra.gr')]).next()
```

Out[142]: ('news247.gr', 'contra.gr', 3.9275702562794277)

In [143]:

```python
def feature_extraction(edge):
    """The function returns the feature vector of the edge
    """
    src, dst = edge
    f_vector = []
    f_vector.append(text_similarity(src, dst))
    f_vector.append(len(set(G.neighbors(src)).intersection(G.neighbors(dst))))
    f_vector.append(second_neighbors(src, dst))
    f_vector.append(G.out_degree(src))
    f_vector.append(G.in_degree(dst))
    f_vector.append(pagerank[src])
    f_vector.append(pagerank[dst])
    f_vector.append(eigenvector[src])
    f_vector.append(eigenvector[dst])
    f_vector.append(betweeness[src])
    f_vector.append(betweeness[dst])
    f_vector.append(closeness[src])
    f_vector.append(closeness[dst])
    f_vector.append(katz[src])
    f_vector.append(katz[dst])
    f_vector.append(core_number[src])
    f_vector.append(core_number[dst])
    f_vector.append(triangles[src])
    f_vector.append(triangles[dst])
    f_vector.append(simrank[domain_number[src], domain_number[dst]])
    f_vector.append(nx.adamic_adar_index(un_g, [(src, dst)]).next()[2])
    f_vector.append(nx.jaccard_coefficient(un_g, [(src, dst)]).next()[2])
    f_vector.append(nx.preferential_attachment(un_g, [(src, dst)]).next()[2])
    f_vector.append(nx.resource_allocation_index(un_g, [(src, dst)]).next()[2])
    f_vector.append(partition_check(src, dst))
    f_vector.append(partition_common(src, dst))
    if G.has_edge(dst,src):
        f_vector.append(1)
    else:
        f_vector.append(0)
    return f_vector

feature_names = ["text","#common_neighbors","#_of_second_neighbors", "G.out_degree(src)","G.in_degree(dst)",
                 "pagerank[src]","pagerank[dst]","eigenvector[src]","eigenvector[dst]","betweeness[src]",
                 "betweeness[src]","betweeness]dst","closeness[src]","closeness[dst]","katz[src]","katz[dst]",
                 "core_number[src]","core_number[dst]","triangles[src]","triangles[dst]","simrank", "adamic_adar",
                 "jaccard_coefficient","preferential_attachment","resource_allocation_index",
                 "partition_check", "opposite_edge"]

# def feature_extraction(edge):
#     src, dst = edge
#     f_vector = []
#     f_vector.append(text_similarity(src, dst))
#     f_vector.append(G.out_degree(src))
#     f_vector.append(G.in_degree(dst))
#     f_vector.append(pagerank[src])
#     f_vector.append(eigenvector[src])
#     f_vector.append(eigenvector[dst])
#     f_vector.append(pagerank[dst])
#     f_vector.append(betweeness[src])
#     f_vector.append(betweeness[dst])
#     f_vector.append(closeness[dst])
#     f_vector.append(closeness[src])
#     f_vector.append(adamic_adar(src, dst))
#     f_vector.append(second_neighbors(src, dst))
#     f_vector.append(partition_check(src, dst))
#     if G.has_edge(dst,src):
#         f_vector.append(1)
#     else:
#         f_vector.append(0)
#     return f_vector


# def feature_extraction(edge):
#     np.random.seed(seed)
#     src, dst = edge
#     feature_names = ["text", "second_neighbors","G.out_degree(src)","G.in_degree(dst)","pagerank[src]"
#                      , "eigenvector[dst]","betweeness[src]","closeness[dst]","preferential_attachment",
#             "resource_allocation_inde","partition_check"]
#     f_vector = []
#     f_vector.append(text_similarity(src, dst))
#     #f_vector.append(len(set(G.neighbors(src)).intersection(G.neighbors(dst))))
#     f_vector.append(second_neighbors(src, dst))
#     f_vector.append(G.out_degree(src))
#     f_vector.append(G.in_degree(dst))
#     f_vector.append(pagerank[src])
#     #f_vector.append(pagerank[dst])
#     #f_vector.append(eigenvector[src])
#     f_vector.append(eigenvector[dst])
#     f_vector.append(betweeness[src])
#     #f_vector.append(betweeness[dst])
#     f_vector.append(closeness[dst])
#     #f_vector.append(closeness[src])
#     #f_vector.append(adamic_adar2(src, dst))
#     #f_vector.append(nx.adamic_adar_index(un_g, [(src, dst)]).next()[2])
#     #f_vector.append(nx.jaccard_coefficient(un_g, [(src, dst)]).next()[2])
#     f_vector.append(nx.preferential_attachment(un_g, [(src, dst)]).next()[2])
#     f_vector.append(nx.resource_allocation_index(un_g, [(src, dst)]).next()[2])
#     f_vector.append(partition_check(src, dst))
#     #return np.random.choice(f_vector,num),seed, names
```

In [144]:
```
seed = np.random.randint(1,4000000)
```

In [145]:
```

#vector,seed,names = feature_extraction(('news247.gr','contra.gr'),5 ,seed)
vector = feature_extraction(('news247.gr','contra.gr'))
if len(feature_names) != len(vector):
    raise Exception
print vector
```

```
[0.47883921986362765, 8, 15, 12, 10, 0.004009557602747434, 0.0016805343926659456, 0.2705070374673354, 0.31052207260
1534, 0.00022283606919962717, 2.1252247833905517e-05, 0.006550802139037433, 0.005404411764705882, 0.092703848650045
54, 0.07013002549244705, 10, 10, 48, 46, 0.0, 3.9275702562794277, 0.2702702702702703, 462, 0.8064976689976691, 1, 1
9, 0]
```

In [146]:
```
X_train = []
y_train = []
for edge in edges:
    X_train.append(feature_extraction(edge))
    y_train.append(1)
for edge in non_existent_edges:
    X_train.append(feature_extraction(edge))
    y_train.append(0)
```

In [147]:
```
# from sklearn.model_selection import train_test_split
# X_train, X_predict, y_train, y_test = train_test_split( X_train, y_train, test_size=0.10, random_state=42)
```

In [149]:
```
X_predict = []
for edge in non_edges:
    X_predict.append(feature_extraction(edge))
```

In [150]:
```
X_train_n = np.array(X_train)
y_train_n = np.array(y_train)
X_predict_n = np.array(X_predict)
```

In [151]:
```
# scale features
from sklearn import preprocessing
# from sklearn import preprocessing
X_train_scaled = preprocessing.scale(X_train_n)
X_predict_scaled = preprocessing.scale(X_predict_n)
```

In [ ]:
```
# PCA didnt work
# from sklearn.decomposition import PCA
# pca = PCA(n_components=10)
# X_train_n = pca.fit_transform(X_train)
# X_predict_n = pca.fit_transform(X_predict)
```

In [154]:
```
print X_train_n.shape
print y_train_n.shape
print X_predict_n.shape
```

```
(23443, 27)
(23443,)
(4140197, 27)
```

Feature Selection

There are four approaches that can be followed for feature selection:

1. Do manual feature selection using the code below. We print some statistics about each feature to see if some of them have low variance. Afterwards we also calculate the pearson correlation between each pair of features and choose to remove one of the features in each pair that shows a high correlation. High correlation between two features can make our model really unstable.
2. Use the feature_selection package available in scikit learn to do feature selection.
3. Use a model that provides feature importance (eg Random Forests).
4. Do no feature selection and use all features in a neural network of a tree based model

We experiment with all the choices we have, but doing feature selection is important and produces a more robust model than using all avalable features.

In [ ]:

```
from scipy import stats
from tabulate import tabulate


#print tabulate(range(0,18), feature_names, tablefmt="grid")
feats = {}
for i, name in enumerate(feature_names):
    feats[name]={}
    feats[name]["mean"]=stats.describe(X_train_n[:,i]).mean
    feats[name]["min_max"]=stats.describe(X_train_n[:,i]).minmax
    feats[name]["variance"]=stats.describe(X_train_n[:,i]).variance

for name, _dict in feats.iteritems():
    print name, tabulate(_dict.items())

from itertools import combinations
f_combs = list(combinations(range(len(feature_names)), 2))

from scipy.stats import pearsonr
pearson_corr = np.zeros((len(f_combs),len(f_combs)))
for f1, f2 in f_combs:
    pearson_corr[f1, f2] = pearsonr(X_train_n[:2683,f1], X_train_n[:2683,f2])[0]
    pearson_corr[f2, f1] = pearson_corr[f1, f2]




#for f1, f2 in f_combs:
    #print feature_names[f1], feature_names[f2], pearson_corr[f1, f2]
for i, name in enumerate(feature_names):
    print name, feature_names[np.argmax(pearson_corr[i, :])] ,pearson_corr[i,np.argmax(pearson_corr[i, :])]
```

In [ ]:

```
# we save the numpy arrays to be able to run classification algorithms
# again without running alla the above processes
np.save("cache/X_train.npy", X_train_n)
np.save("cache/y_train.npy", y_train_n)
np.save("cache/X_test.npy", X_predict_n)
```

Classification

For classication we test several cases. At first we use logistic regression, SVM and a neural network classifier.
As we have rather small data opposed to the candidate space (2683 true edges opposed to ~4m edges) we have to be very careful with regularization because most models tend to overfit the training data.
Afterward we experiment with Random Forests as well as ensemble methods such as Gradient boosting, XGBoost which are methods that have gained much attendance lately, especially due to their high performance in Kaggle contests.
However such models require very good tuning because of the high parameter space they have, so we perform Grid Search to find the optimal parameters for the Random Forest model as well as XGBoost with all the candidate features. Unfortunately because of the computational complexity of such a task we did not perform Grid Search to models with different features. In the end we also run a voting classifier with the best tuning for XGBoost and Random Forest.
We want to use classification algorithms that implement the predict_proba method, because we are interested in the probability that an edge will exist in our graph. Because of the small dataset and the overfitting that happens we must balance the classifiers uncertainty.

The highest (reproducible) score (10,15%) was obtained using XGBoost and the following feature vector function:

def feature_extraction(edge):
src, dst = edge
f_vector = []
f_vector.append(text_similarity(src, dst))
f_vector.append(pagerank[src])
f_vector.append(eigenvector[src])
f_vector.append(eigenvector[dst])
f_vector.append(pagerank[dst])
f_vector.append(betweeness[src])
f_vector.append(betweeness[dst])
f_vector.append(closeness[dst])
f_vector.append(closeness[src])
f_vector.append(adamic_adar(src, dst))
f_vector.append(partition_check(src, dst))
return f_vector

We also scored a 10.31% but it was with a Random Forest without saving the random seed.

In [ ]:

```
# if we have not ran everything above we can just run this to load the Train/Test data.
X_train_n = np.load('cache/X_train.npy')
y_train_n = np.load('cache/y_train.npy')
X_predict_n = np.load('cache/X_test.npy')
```

In [ ]:
```
from sklearn.linear_model import LogisticRegression
reg = LogisticRegression(C= 0.00001)
reg.fit(X_train_scaled, y_train_n)
pred_reg = reg.predict_proba(X_predict_scaled)
probs_reg = []
for t in pred_reg:
    probs_reg.append(t[1])
indices = sorted(range(len(probs_reg)), key=lambda k: probs_reg[k])[::-1][:453]
print [probs_reg[indices[i]] for i in range(len(indices))]
predicted_edges_reg = []
for i in range(len(indices)):
    predicted_edges_reg.append(non_edges[indices[i]])

with open('predicted_edges_logReg.txt', 'w') as fp:
    for site_1, site_2 in predicted_edges_reg:
        fp.write(site_1+'\t'+site_2+'\n')
```

In [ ]:
```
from sklearn.svm import SVC
svm = SVC(probability=True, C=0.00001)
svm.fit(X_train_scaled, y_train_n)
pred_svm = svm.predict_proba(X_predict_scaled)
probs_svm = []
for t in pred_svm:
    probs_svm.append(t[1])
indices = sorted(range(len(probs_svm)), key=lambda k: probs_svm[k])[::-1][:453]
print [probs_svm[indices[i]] for i in range(len(indices))]
predicted_edges_svm = []
for i in range(len(indices)):
    predicted_edges_svm.append(non_edges[indices[i]])

with open('predicted_edges_svm.txt', 'w') as fp:
    for site_1, site_2 in predicted_edges_svm:
        fp.write(site_1+'\t'+site_2+'\n')
```

In [100]:

```
from sklearn.neural_network import MLPClassifier
mlp = MLPClassifier(activation = 'logistic',solver='lbfgs', alpha=1e-5, hidden_layer_sizes=(80,40), random_state=1)

mlp.fit(X_train_scaled, y_train_n)

pred_mlp = mlp.predict_proba(X_predict_scaled)

probs_mlp = []
for t in pred_mlp:
    probs_mlp.append(t[1])
indices = sorted(range(len(probs_mlp)), key=lambda k: probs_mlp[k])[::-1][:453]
print [probs_mlp[indices[i]] for i in range(len(indices))]

predicted_edges_mlp = []
for i in range(len(indices)):
    predicted_edges_mlp.append(non_edges[indices[i]])

with open('predicted_edges_mlp.txt', 'w') as fp:
    for site_1, site_2 in predicted_edges_mlp:
        fp.write(site_1+'\t'+site_2+'\n')
```

```
[0.99999999927485139, 0.99999999923454008, 0.99999999922941729, 0.99999999922219884, 0.99999999922191551, 0.9999999
9922025284, 0.99999999921885618, 0.99999999921776261, 0.99999999921626448, 0.99999999921249527, 0.99999999921225124
, 0.9999999992119053, 0.99999999920858462, 0.99999999920089411, 0.99999999919887639, 0.99999999919246996, 0.9999999
9919198879, 0.99999999919008742, 0.99999999918713622, 0.99999999918668458, 0.99999999918657068, 0.99999999918637839
, 0.99999999918403204, 0.99999999918021465, 0.99999999917980809, 0.99999999917965976, 0.9999999991769255, 0.9999999
9917570248, 0.99999999917507876, 0.99999999917405935, 0.99999999917349447, 0.99999999917132421, 0.99999999916726945
, 0.99999999916467464, 0.99999999916465998, 0.9999999991626165, 0.99999999916258098, 0.99999999916062987, 0.9999999
9915708182, 0.99999999915371074, 0.99999999915325377, 0.9999999991528572, 0.99999999915073023, 0.99999999914845694,
0.99999999914783122, 0.99999999914693705, 0.99999999914402227, 0.99999999914382909, 0.99999999914382753, 0.99999999
914308724, 0.9999999991401527, 0.99999999914013538, 0.99999999913837923, 0.99999999913836057, 0.99999999913672899,
0.99999999913636239, 0.9999999991352575, 0.99999999913418969, 0.99999999913388371, 0.99999999913296489, 0.999999999
1313635, 0.99999999912978343, 0.99999999912909132, 0.99999999912791226, 0.99999999912784765, 0.99999999912752235, 0
.99999999912744131, 0.99999999912734228, 0.99999999912458848, 0.99999999912370163, 0.99999999912312409, 0.999999999
12238269, 0.99999999912169391, 0.99999999912113724, 0.99999999911196642, 0.9999999991194819, 0.99999999911751436, 0.
99999999911749238, 0.9999999991169275, 0.99999999911512782, 0.99999999911428183, 0.99999999911327886, 0.99999999911
204518, 0.99999999911156889, 0.99999999911064785, 0.99999999911034121, 0.9999999991091737, 0.99999999910845117, 0.9
9999999910661352, 0.99999999910567072, 0.99999999910547155, 0.99999999910449611, 0.9999999991033699, 0.999999999101
35595, 0.99999999910135373, 0.99999999909969906, 0.99999999909934822, 0.99999999909828841, 0.99999999909817894, 0.9
999999990981554, 0.99999999909801707, 0.99999999909784876, 0.99999999909678539, 0.99999999909673143, 0.999999999096
15056, 0.99999999909401982, 0.99999999909128734, 0.99999999909089943, 0.99999999908949788, 0.99999999908871984, 0.9
9999999908768689, 0.99999999908765513, 0.99999999908728343, 0.99999999908689263, 0.99999999908680337, 0.99999999908
625425, 0.99999999908413528, 0.99999999908315274, 0.99999999908210513, 0.99999999908147519, 0.99999999908078929, 0.
99999999908056747, 0.99999999907960468, 0.99999999907950499, 0.99999999907921744, 0.99999999907864012, 0.9999999990
7652959, 0.99999999907631842, 0.99999999907579085, 0.99999999907473258, 0.99999999907470105, 0.99999999907396053, 0
.99999999907395098, 0.99999999907331216, 0.99999999907202208, 0.99999999907047354, 0.99999999906995285, 0.999999999
0698496, 0.99999999906923787, 0.99999999906872072, 0.99999999906868298, 0.9999999990677142, 0.99999999906723214, 0.
9999999990671562, 0.99999999906680781, 0.99999999906674941, 0.99999999906665371, 0.99999999906617565, 0.99999999906
475434, 0.99999999906458736, 0.99999999906409798, 0.99999999906060721, 0.99999999905952608, 0.99999999905928871, 0.
99999999905619052, 0.9999999990547277, 0.99999999905398052, 0.9999999990538857, 0.99999999905363146, 0.999999999052
36714, 0.99999999905219261, 0.99999999905027948, 0.99999999905004433, 0.99999999904797576, 0.99999999904649783, 0.9
9999999904575221, 0.99999999904563897, 0.99999999904549797, 0.99999999904539738, 0.99999999904310388, 0.99999999904
259429, 0.99999999904235071, 0.99999999904125736, 0.99999999904069137, 0.999999999040184, 0.99999999903986581, 0.99
999999903789005, 0.99999999903771952, 0.99999999903627734, 0.99999999903609149, 0.99999999903463843, 0.999999999033
94365, 0.9999999990329782, 0.99999999903152847, 0.99999999903149117, 0.99999999903071712, 0.99999999903033632, 0.99
999999902986514, 0.99999999902983649, 0.99999999902954295, 0.99999999902952363, 0.9999999990292836, 0.999999999286
0437, 0.99999999902858727, 0.99999999902684822, 0.99999999902557168, 0.999999999025035, 0.9999999990241899, 0.99999
999902372316, 0.99999999902299397, 0.9999999990225712, 0.99999999901877779, 0.99999999901819181, 0.9999999990180648
, 0.99999999901769354, 0.99999999901742398, 0.99999999901712444, 0.99999999901703274, 0.99999999901632064, 0.999999
9990161228, 0.99999999901605776, 0.99999999901586989, 0.99999999901568204, 0.99999999901371495, 0.99999999901353509
, 0.99999999901343362, 0.99999999901203362, 0.9999999990117121, 0.99999999901154135, 0.99999999901098113, 0.9999999
9901086545, 0.99999999901048864, 0.99999999901039138, 0.99999999900779102, 0.99999999900694503, 0.99999999900625047
, 0.99999999900608549, 0.99999999900467995, 0.99999999900465331, 0.99999999900456116, 0.99999999900430248, 0.999999
99900417835, 0.99999999900377468, 0.99999999900324554, 0.99999999900317271, 0.99999999900290515, 0.9999999990006815
9, 0.99999999900047509, 0.9999999990004742, 0.99999999900034942, 0.99999999900034098, 0.99999999899810277, 0.999999
99899620562, 0.99999999899565695, 0.99999999899544334, 0.99999999899535141, 0.99999999899528857, 0.9999999989946650
7, 0.99999999899415282, 0.99999999899391656, 0.99999999899375136, 0.99999999899363368, 0.99999999899344627, 0.99999
999899327818, 0.99999999899295067, 0.99999999899202852, 0.99999999899039382, 0.99999999899029768, 0.999999998989251
18, 0.99999999898901359, 0.99999999898835101, 0.99999999898813807, 0.99999999898768421, 0.99999999898663749, 0.9999
9999898612368, 0.99999999898572534, 0.99999999898487335, 0.99999999898431269, 0.99999999898422365, 0.99999999898413
128, 0.99999999898369984, 0.99999999898252523, 0.999999998981973, 0.99999999898169678, 0.9999999989812598, 0.999999
9989811692, 0.99999999898095338, 0.99999999898049885, 0.99999999898018577, 0.99999999897950009, 0.99999999897863523
, 0.99999999897823222, 0.9999999989776529, 0.99999999897750613, 0.99999999897688663, 0.999999998977671 61, 0.999999
897662484, 0.99999999897654757, 0.99999999897640679, 0.9999999989763162, 0.99999999897625425, 0.99999999897453429,
0.99999999897313163, 0.99999999897145408, 0.99999999897117275, 0.99999999897066805, 0.99999999897052883, 0.99999999
897017977, 0.99999999896999725, 0.99999999896887481, 0.99999999896879688, 0.99999999896868719, 0.99999999896826974,
0.99999999896800817, 0.99999999896792091, 0.99999999896779923, 0.99999999896758518, 0.99999999896757719, 0.99999999
896713598, 0.99999999896707203, 0.99999999896637282, 0.99999999896510339, 0.99999999896483582, 0.99999999896483138,
0.99999999896437131, 0.99999999896340452, 0.99999999896288982, 0.99999999896204872, 0.99999999896144054, 0.99999999
896126845, 0.99999999895970371, 0.99999999895965686, 0.999999998959334, 0.99999999895914948, 0.99999999895903691, 0
.99999999895895675, 0.99999999895723746, 0.99999999895721325, 0.99999999895663105, 0.99999999895628666, 0.999999998
95605463, 0.99999999895453362, 0.99999999895450009, 0.9999999989543642, 0.99999999895420588, 0.99999999895406222, 0
.99999999895370006, 0.9999999989534567, 0.9999999989529198, 0.99999999895252456, 0.99999999895249392, 0.99999999895
10171, 0.99999999894985381, 0.99999999894894764, 0.99999999894809299, 0.99999999894794511, 0.99999999894779101, 0.9
9999999894683356, 0.99999999894641078, 0.99999999894556169, 0.99999999894418212, 0.99999999894367964, 0.99999999894
315317, 0.99999999894235803, 0.99999999894233027, 0.99999999894162372, 0.9999999989413455, 0.99999999894062253, 0.9
9999999894054148, 0.99999999894017555, 0.99999999894008385, 0.99999999893989244, 0.9999999989386259, 0.999999998937
98797, 0.99999999893792957, 0.99999999893778435, 0.99999999893746438, 0.99999999893652602, 0.99999999893565228, 0.9
999999989348316, 0.99999999893456004, 0.99999999893417013, 0.99999999893402958, 0.99999999893318336, 0.999999998932
39, 0.99999999893236602, 0.99999999893127289, 0.99999999893105729, 0.9999999989304118, 0.99999999893009028, 0.99999
999892980274, 0.99999999892958247, 0.99999999892884595, 0.99999999892854308, 0.99999999892825997, 0.999999998927682
43, 0.99999999892744484, 0.99999999892738622, 0.99999999892709157, 0.99999999892662195, 0.99999999892655089, 0.9999
9999892557412, 0.99999999892477787, 0.99999999892366054, 0.99999999892317915, 0.99999999892310254, 0.99999999892291
713, 0.99999999892270264, 0.99999999892156022, 0.99999999892139124, 0.99999999892079305, 0.99999999892045821, 0.999
99999892016223, 0.99999999891980695, 0.99999999891951163, 0.99999999891772529, 0.99999999891726565, 0.9999999989171
7906, 0.99999999891601266, 0.999999998915998, 0.99999999891566005, 0.99999999891556679, 0.99999999891440683, 0.9999
9999891408931, 0.99999999891387459, 0.99999999891284785, 0.99999999891241487, 0.99999999891215618, 0.99999999891165
903, 0.999999998911272, 0.99999999891106706, 0.99999999891052682, 0.99999999890975011, 0.99999999890905378, 0.99999
999890775415, 0.99999999890691438, 0.99999999890595248, 0.99999999890524149, 0.99999999890376201, 0.999999998903422
73, 0.99999999890292757, 0.99999999890153601, 0.9999999988996866, 0.99999999889919855, 0.99999999889842406, 0.99999
999889806368, 0.99999999889745506, 0.99999999889688862, 0.99999999889629221, 0.9999999988949364, 0.9999999988945791
3, 0.99999999889442148, 0.99999999889429736, 0.99999999889369406, 0.99999999889367808, 0.99999999889355951, 0.99999
999889351554, 0.99999999889302416, 0.99999999889056546, 0.99999999888897362, 0.99999999888867275, 0.999999998887899
81, 0.99999999888771818]
```

In [16]:
```python
import numpy as np
import random
X_train_n = np.load('cache/X_train.npy')
y_train_n = np.load('cache/y_train.npy')
X_predict_n = np.load('cache/X_test.npy')
```

In [25]:
```
from sklearn.ensemble import RandomForestClassifier
np.random.seed(10)
seed = np.random.randint(1,4000000)
print seed
#{'bootstrap': False, 'min_samples_leaf': 3, 'min_samples_split': 10, 'criterion': 'gini',
                #'max_features': 4, 'max_depth': 8}
    #'bootstrap': True, 'min_samples_leaf': 3, 'min_samples_split': 2, 'criterion': 'entropy',
    #'max_features': 8, 'max_depth': 8}
#forest = RandomForestClassifier(random_state=seed,n_estimators=80, min_samples_split=10,
                                #max_features=4, max_depth=8, min_samples_leaf=3,criterion='gini')
# Parameters: {'bootstrap': True, 'min_samples_leaf': 3, 'min_samples_split': 2, 'criterion': 'gini',
#              'max_features': 3, 'max_depth': 8}

forest = RandomForestClassifier(random_state=seed, n_estimators=120, min_samples_split=3, min_samples_leaf=3,
                                bootstrap=False,  max_features=3, max_depth=8, criterion='gini')

forest.fit(X_train_n, y_train_n)
pred_forest = forest.predict_proba(X_predict_n)
probs_forest = []
for t in pred_forest:
    probs_forest.append(t[1])
indices = sorted(range(len(probs_forest)), key=lambda k: probs_forest[k])[::-1][:453]
predicted_edges_forest = []
for i in range(len(indices)):
        predicted_edges_forest.append(non_edges[indices[i]])
print [probs_forest[indices[i]] for i in range(len(indices))]
with open('predicted_edges_forest.txt', 'w') as fp:
    for site_1, site_2 in predicted_edges_forest:
        fp.write(site_1+'\t'+site_2+'\n')
f_ind = np.argsort(forest.feature_importances_)[::-1]
for i in f_ind:
    print "\n",feature_names[i],forest.feature_importances_[i]
```

```
3491082
[0.99933635309435287, 0.99933635309435287, 0.99933635309435287, 0.99933635309435287, 0.99918575068471438, 0.9990489
967725138, 0.99902188768554778, 0.99889839436287531, 0.99889839436287531, 0.99889839436287531, 0.99889839436287531,
0.99737104507951657, 0.99709638124103361, 0.9959911749496464, 0.99359901599083689, 0.99359901599083689, 0.992701580
09340104, 0.99210241735138094, 0.99108417370037893, 0.98929045402356885, 0.98885593019822271, 0.98885593019822271,
0.98877148255086689, 0.98842839731540322, 0.98761034276397131, 0.98756902022533233, 0.98747243709792687, 0.98650915
984931198, 0.98628771927909342, 0.9859873767905345, 0.98570822654452372, 0.98495087237278589, 0.98440998479999453,
0.98438379467722759, 0.98373502640826416, 0.98363666804452732, 0.98355555648459425, 0.98318388057941275, 0.98301446
176621554, 0.9828521801632869, 0.98280763304734475, 0.98270658495145824, 0.98244347278543664, 0.98214918687809016,
0.98173436272923587, 0.9815280280584997, 0.98143290799651972, 0.98132768485137523, 0.98121108963345494, 0.981084847
85567485, 0.98098796938866595, 0.980947007056859, 0.98085162334572O4, 0.98029576243666461, 0.98027509422630688, 0.9
8009127380804706, 0.98003897107097948, 0.9799863113499202, 0.97985476368587132, 0.97984240390685351, 0.979755180955
00369, 0.97973927179706344, 0.97973397331239742, 0.97964518074563167, 0.97963788335722979, 0.97955465799428831, 0.9
7954203333448642, 0.97947741348605766, 0.97934228185094319, 0.9793099828475047, 0.97928322058256112, 0.979099539054
04512, 0.97900151452234152, 0.97819352566265749, 0.97807787572487215, 0.97750959017110617, 0.97718146488103164, 0.9
7710513325459558, 0.97704595153163076, 0.97698434867480966, 0.97680811242460386, 0.97680228287131787, 0.97661999464
039484, 0.97655920760865811, 0.97639406518412275, 0.97636382421075363, 0.97617654037149826, 0.9760145225763438, 0.9
759792102173066, 0.97590447527535507, 0.97565286852881039, 0.97559743938814603, 0.97558897418674451, 0.975586796808
41653, 0.97541723232384758, 0.97492554772126827, 0.97491294925576888, 0.97467160401958541, 0.97466191975514505, 0.9
745961851165853, 0.97454787416236988, 0.97447194839055229, 0.9744420820470967, 0.97443310456702548, 0.9742794900392
0322, 0.97416620452556668, 0.97412729622985805, 0.97408578691686498, 0.97404830938162612, 0.97403676090148938, 0.97
39603247892652, 0.97331527091021386, 0.97323055120568014, 0.97306138799504061, 0.97278216461698541, 0.9721298135294
7037, 0.97211734018044493, 0.97202817731105673, 0.97197120061339226, 0.97164995074925808, 0.97159393177804643, 0.97
139724438298836, 0.97105400022023403, 0.9709825418336514, 0.97091000334693633, 0.97084849825364572, 0.9707871276722
1498, 0.97067651586683934, 0.97065182926492555, 0.97058679680841642, 0.97036190891894902, 0.97028417284924362, 0.97
000669287827346, 0.96997774023240702, 0.96938488301163683, 0.96833801684032539, 0.96829716585124159, 0.968203287963
73245, 0.96804054992345712, 0.96791688662777753, 0.96788221683841036, 0.96759661977613232, 0.96751380914792795, 0.9
6738636444555959, 0.96735846937821213, 0.96734302413280959, 0.96729215865228391, 0.96711380730468732, 0.96687561544
202516, 0.96659581012005924, 0.9665057836729497, 0.96648287163622959, 0.96618724131291966, 0.96596611455642667, 0.9
657859630817478, 0.96566249183194974, 0.96552138944589194, 0.96527265377852312, 0.96526295459894873, 0.965232609637
13764, 0.96492365119875279, 0.96478249031998842, 0.96474409609134304, 0.96473582364180777, 0.96466836525463895, 0.9
6466524876312709, 0.96438363542889505, 0.96429210142545796, 0.96421593662248795, 0.96408035087999966, 0.96363951130
049808, 0.96307254681105803, 0.96286835390705139, 0.96276174444518747, 0.9626641422233374, 0.96262599501111801, 0.9
6257688109502426, 0.96256623808671204, 0.96255576113902186, 0.96239426838914, 0.96237402631561197, 0.96221640522738
86, 0.96216310148221029, 0.96204737887990466, 0.9614966187669195, 0.9614820788491496, 0.96137971642681663, 0.961231
04296064299, 0.96120551379985553, 0.96112769200329606, 0.96092665791909115, 0.96040941836138116, 0.9604094183613811
6, 0.96026791562461911, 0.96012291577403075, 0.95988445080549145, 0.95988067082755646, 0.95981082308127585, 0.95977
752247325232, 0.95973956208832611, 0.95972371790964872, 0.95964615259143882, 0.95961690684893874, 0.959599309769524
68, 0.95958867630979761, 0.9594532675466042, 0.95940503577261682, 0.9593923828291685, 0.95938532370443264, 0.959313
51787011698, 0.95924906968137902, 0.95910046879239941, 0.95904573440856578, 0.95902020387105247, 0.9588927856020660
3, 0.95871785345270755, 0.95863359152380812, 0.9585216300080398, 0.95849142792173569, 0.95833340455345561, 0.958322
09833783133, 0.95827836151844581, 0.95819847987536055, 0.95819512496123482, 0.95795262132235892, 0.9579016541418111
7, 0.95782038968856253, 0.95744873979876743, 0.95741097712156875, 0.95709808194550694, 0.95698300108461398, 0.95695
918753928866, 0.95695052950442294, 0.95685422169843948, 0.9567296076435442, 0.956550795025634, 0.95649708432220804,
0.95643083619700076, 0.956418706699393, 0.9564059858466909, 0.95634230552814659, 0.95626923787413332, 0.95626610537
718215, 0.95616894376686268, 0.95615348246984833, 0.9561373805583846, 0.95613182593341139, 0.95609055698184697, 0.9
5587814854748099, 0.95586830321967464, 0.95577823868870848, 0.95574294801413917, 0.95567810060710923, 0.95564475681
194372, 0.95555102087397392, 0.95540489065151912, 0.95519720174405798, 0.95519389724260495, 0.95518059180259562, 0.
95517215278944734, 0.95510333503417955, 0.95501231352977356, 0.95498987592901419, 0.95489654382357458, 0.9548776857
8915599, 0.9548728369045405, 0.95484364274648947, 0.95483111584695002, 0.95482741907602164, 0.95481503251068023, 0.
95440894916973484, 0.95424003500737098, 0.95421861656946505, 0.95411837186391302, 0.95408660275332768, 0.9540831116
0204262, 0.95407609129217286, 0.95390117669589436, 0.95381902100580662, 0.95381638266123236, 0.95378885405011771, 0
.95374770285791632, 0.95349607631683664, 0.9534755351069838, 0.95338723941792913, 0.95334655308960647, 0.9533454207
2884532, 0.95323238007208966, 0.95322733114207814, 0.95319984048613349, 0.95319822349946259, 0.95309273988255505, 0
.95297885168993457, 0.95293074526711896, 0.95288215268885679, 0.95280897651972429, 0.95276977678695352, 0.952752966
81294819, 0.95273172647604731, 0.95260015758492578, 0.95249457003281701, 0.95248515888068008, 0.95248301919131106,
0.95229140538347434, 0.95215885051829863, 0.95213823884687676, 0.95206370081239144, 0.95180946082499907, 0.95152085
652824314, 0.95148531791883595, 0.95145210397061974, 0.95136616848295263, 0.95135090315993764, 0.9512063872295009,
0.95116678702623525, 0.95110938207266227, 0.95099349235418973, 0.95080602702999562, 0.95068302615559941, 0.95061038
85759748, 0.95060540983315389, 0.95060400148546598, 0.95053925852329468, 0.95044196645387102, 0.95032108249293867,
0.95026560670362725, 0.95022167242975952, 0.95021197016582526, 0.9502033980145671, 0.95019822568132084, 0.950195342
66065753, 0.95019523041493914, 0.94988251884543795, 0.94983613675093026, 0.94971674525387217, 0.94968778688416222,
0.94956896064100005, 0.94937832175491954, 0.94928118800418604, 0.94924557755449424, 0.94923071190184527, 0.94914428
006018958, 0.94903981397030712, 0.94901776029196394, 0.94890942457395178, 0.94890744757011047, 0.94889819629531769,
0.94879416577097797, 0.94856829396966325, 0.94848429348541252, 0.94829668970508352, 0.94819077404163599, 0.94808469
915603877, 0.94807702335585942, 0.94804597334277041, 0.94801919457645811, 0.94800774022487144, 0.9478585505361441,
0.94782798579160288, 0.9476603932419061, 0.94758190673616494, 0.94746656037554955, 0.94745911149481743, 0.947377817
77918717, 0.94736384328177237, 0.94734870435529583, 0.9473247373850523, 0.94726349334145277, 0.94703907432109236, 0
.94698905743672579, 0.94697923258734917, 0.94693121642420164, 0.94692913077795748, 0.94690383738988548, 0.946902428
12247916, 0.94688723928124763, 0.94688606611710313, 0.94677535593684337, 0.94664364007718427, 0.9464877487687855, 0
.94639828168722306, 0.94635296079612241, 0.94619362529095852, 0.94609428814327112, 0.9460781439925593, 0.9460394544
1201753, 0.94601884272087611, 0.94588291077462494, 0.9458720970950949, 0.94584794192552968, 0.94584588024730065, 0.
9457946758419804, 0.94578086353605628, 0.9457753162030843, 0.94574129277248931, 0.94570499110788531, 0.945670641043
00214, 0.94565200248900172, 0.9454849610228071, 0.94541483641558444, 0.94539550761687019, 0.94526507273728688, 0.94
524551087750019, 0.94515814270869791, 0.9451208649635553, 0.94511532995612879, 0.94507491978377645, 0.9450475923475
3499, 0.94502328777848066, 0.94502328777848066, 0.94501737501107996, 0.94500402673264505, 0.94497685162969547, 0.94
492487162770711, 0.94487182913839307, 0.94486958214559447, 0.9448403373134705, 0.9448223265604373, 0.94462194052958
859, 0.94459740681836302, 0.94452709248034594, 0.94452284137444442, 0.94452284137444442, 0.94446684013432913, 0.944
41021213988663, 0.94432508062319065, 0.94422519300013097, 0.94415102504045378, 0.94407759279683934, 0.9440649438025
1504, 0.94395307778938797, 0.94373259249381991, 0.94372522341139253, 0.94371627204776709, 0.94369184382013038, 0.94
363729892376003, 0.94357740454661665, 0.94357740454661665, 0.94357740454661665, 0.94357096537317331, 0.943525600269
91594, 0.94349889320116098, 0.94348576681613772, 0.94348106382325747, 0.94340103127258124, 0.94339902632418893, 0.9
4338487129525705, 0.94337776091677161, 0.9433774260172817, 0.94335026988947157, 0.94333937084775732, 0.943325518692
04628, 0.94324001006366132]
partition_check 0.391601768122
resource_allocation_index 0.245702495214
jaccard_coefficient 0.0584018261785
G.out_degree(src) 0.0502279324327
G.in_degree(dst) 0.0443922104012
katz[src] 0.0439536551598
betweeness]dst 0.0389682085841
pagerank[dst] 0.0277850050799
katz[dst] 0.0154846437819
preferential_attachment 0.0143927574732
core_number[src] 0.0126586048192
simrank 0.00767806136126
```

In [ ]:

```
import xgboost
xgb = xgboost.XGBClassifier()
xgb.fit(X_train_n, y_train_n)
pred_xgb = xgb.predict_proba(X_predict_n)

f_ind = np.argsort(xgb.feature_importances_)[::-1]
for i in f_ind:
    print feature_names[i],xgb.feature_importances_[i]

probs_xgb = []
for t in pred_xgb:
    probs_xgb.append(t[1])
indices = sorted(range(len(probs_xgb)), key=lambda k: probs_xgb[k])[::-1][:453]
predicted_edges_xgb = []
counter=0
for i in range(len(indices)):
    predicted_edges_xgb.append(non_edges[indices[i]])
print [probs_xgb[indices[i]] for i in range(len(indices))]
with open('predicted_edges_xgb.txt', 'w') as fp:
    for site_1, site_2 in predicted_edges_xgb:
        fp.write(site_1+'\t'+site_2+'\n')
```

Gradient Boosting

In [28]:

```
class CompatClassifier(RandomForestClassifier):
    def predict(self, X):
        return self.predict_proba(X)[:, 1][:,np.newaxis]
seed = random.randint(1, 400000)
print seed
#278753

base_estimator = CompatClassifier(random_state=3808056, n_estimators=10, min_samples_split=3, min_samples_leaf=3,
                                bootstrap=False,  max_features=3, max_depth=8, criterion='gini')
#min_samples_split=10,
                                #max_features=4, max_depth=8, min_samples_leaf=3,criterion='gini'
from sklearn.ensemble import GradientBoostingClassifier
gbc = GradientBoostingClassifier(random_state=3808056,init=base_estimator,n_estimators=150)

gbc.fit(X_train_n,y_train_n)
preds_gbc = gbc.predict_proba(X_predict_n)
probs_gbc = []
for t in preds_gbc:
    probs_gbc.append(t[1])
indices = sorted(range(len(probs_gbc)), key=lambda k: probs_gbc[k])[::-1][:453]
predicted_edges_gbc = []
for i in range(len(indices)):
        predicted_edges_gbc.append(non_edges[indices[i]])
print [probs_gbc[indices[i]] for i in range(len(indices))]
with open('predicted_edges_gbc.txt', 'w') as fp:
    for site_1, site_2 in predicted_edges_gbc:
        fp.write(site_1+'\t'+site_2+'\n')
f_ind = np.argsort(gbc.feature_importances_)[::-1]
for i in f_ind:
    print "\n",feature_names[i],gbc.feature_importances_[i]
```

231237
[0.99990616443726688, 0.99989693841848748, 0.9998922135588949, 0.99988683938769096, 0.99987787698348329, 0.99987527
473687998, 0.99986852023662576, 0.99986574054235766, 0.99986418456194326, 0.99986177513969532, 0.99986072271133863,
0.99985814661859973, 0.99985128516004773, 0.99984314385779904, 0.99984074948899948, 0.99983943596107994, 0.99983458
566184436, 0.99982975321975842, 0.99981782635897509, 0.99981782635897509, 0.99980833273367997, 0.99980272558311034,
0.99979805495144802, 0.99979786179723296, 0.99979750946688462, 0.99979728275942537, 0.99979706372470112, 0.99979704
778274203, 0.99979115408838937, 0.9997877920283178, 0.99978345264977553, 0.99978236365253526, 0.9997782472990473, 0
.99977650717199418, 0.99977091796561324, 0.99977026501521715, 0.99976628134892487, 0.99976570637735251, 0.999762122
16886828, 0.99975728058394364, 0.99975464400733816, 0.99975206550359208, 0.99975199676534676, 0.99975065085003945,
0.99974980430906424, 0.99974702336200116, 0.99974622879314978, 0.99974430519227464, 0.99974055445966936, 0.99973396
32902642, 0.9997337941815726, 0.99972801740269668, 0.99972176009705793, 0.99972091590271051, 0.99971925021096109, 0
.99971878239691769, 0.99970894489728312, 0.99970793731141883, 0.99970542695326003, 0.99970350938160546, 0.999702706
49280324, 0.99970215296517806, 0.99970159683629567, 0.99970010029054357, 0.99969857275515151, 0.99969622306146544, 0
.99969503723870512, 0.99969472210583044, 0.99969399818891302, 0.99969377267391624, 0.99969174724650955, 0.999690653
69846022, 0.99969006640783742, 0.99968670703348017, 0.99968657727379029, 0.99968497095888875, 0.99968475876546425,
0.99968366007076326, 0.99968262322001966, 0.99968020097827159, 0.99967741368717256, 0.99967701513714013, 0.99967444
98496748, 0.99967394835487211, 0.99967258660288816, 0.99967258660288816, 0.99967258660288816, 0.99967143235733968,
0.99967143235733968, 0.9996713523473757, 0.99967123078654541, 0.99967123078654541, 0.99966758038411474, 0.99966501
16646014, 0.9996641873648926, 0.99966372243160062, 0.99966353183968581, 0.99966301052478157, 0.99966292024903358, 0
.99966114155379349, 0.99966062371980147, 0.99965875064543364, 0.99965813662197756, 0.99965552118175416, 0.999655405
47961718, 0.99965402005159931, 0.99965319743764514, 0.99965200961634215, 0.99963945384692787, 0.99963565924998021,
0.99962967570579675, 0.99962943265662596, 0.99962906891644776, 0.99962569191669937, 0.99962458876999882, 0.99962320
135446747, 0.99962086333217948, 0.99962019476100195, 0.99961730998509257, 0.99961385246647416, 0.99961300049544843,
0.99961253318298904, 0.9996092182789813, 0.99960540823535016, 0.99960360537237603, 0.99959993699165939, 0.999597833
19979545, 0.99959777381443371, 0.99959751466338953, 0.99959751466338953, 0.9995974594133159, 0.9995953147939316, 0.
99959457848779432, 0.99959141560264664, 0.99958622850046808, 0.99958580713955392, 0.99957964418799139, 0.9995786162
7286515, 0.99957409024507848, 0.99957042577542721, 0.99957038979284196, 0.99956954535896869, 0.99956929375520454, 0
.99956928130912348, 0.99956828206573023, 0.99956551624641632, 0.9995588928934046, 0.99955776895796056, 0.9995563291
995182, 0.99955592089224077, 0.99955505482831608, 0.99955430089627351, 0.99955316035792396, 0.99955242547679912, 0.
99955128448824715, 0.99954908590461478, 0.99954566153195434, 0.99954337323277553, 0.99954325637999353, 0.9995405053
7122185, 0.9995391107890379, 0.99953878123122142, 0.99953839154958113, 0.99953791408674675, 0.99953600643315288, 0.
99952818268039256, 0.99952780548216591, 0.99952555431817125, 0.99952522822690759, 0.99952463105682121, 0.9995206446
5501922, 0.99951632184552841, 0.99951495597762552, 0.99951364417661737, 0.99951290928728898, 0.99950956149456727, 0
.99950728414027146, 0.99950451735516421, 0.9995043901345082, 0.99950288546789212, 0.99949812414403461, 0.9994970827
9436889, 0.9994959106033775, 0.99949397373382927, 0.99948903400790845, 0.99948903400790845, 0.99948675052847891, 0.99
947970337448222, 0.99947949852442497, 0.99947195818923962, 0.99947102082181205, 0.99946735758223637, 0.999465506032
67115, 0.99946203446158577, 0.9994585230188493, 0.9994572901419031, 0.99945712418316868, 0.9994545211360516, 0.9994
5452050984274, 0.99945148213481827, 0.99945060589817558, 0.99944667838929691, 0.99944428724962775, 0.99944414212913
268, 0.9994430044717828, 0.99944197192531448, 0.99943856284271682, 0.99943556287658897, 0.99943383696632382, 0.9994
3173314699663, 0.99943135112004289, 0.99943120470492453, 0.99942809459325355, 0.99942768484382993, 0.99942697509616
452, 0.99942697509616452, 0.99942549223313015, 0.9994213526457254, 0.99941913586297471, 0.9994179651024222, 0.99941
659572654895, 0.99941543145276468, 0.9994103212736043, 0.9994079484313978, 0.9994043675047577, 0.99940434779388676,
0.99940243600091579, 0.99940125828119997, 0.99940110565594942, 0.99939914075385672, 0.99939868927628017, 0.999398423
23780248, 0.9993937372588404, 0.99939302511907435, 0.99939198726542977, 0.99938735717293836, 0.99938696937948568, 0
.9993869088248839, 0.99938269425046666, 0.99938228778856664, 0.99938143821055758, 0.99937973919295831, 0.9993795056
9283707, 0.99937756148338208, 0.99937747779162189, 0.99937715595440468, 0.99937687760245952, 0.9993752143447352, 0.
99937486783671314, 0.99937137338430027, 0.99936602188987356, 0.99936602188987356, 0.99936602188987356, 0.9993637862
5510602, 0.99936331205215756, 0.99936255536950491, 0.99936194580556392, 0.99936194580556392, 0.99936109197027345, 0
.99936085516686135, 0.99935785403297872, 0.99935204061367588, 0.99935145648566548, 0.99935141878992484, 0.999349807
25384581, 0.99934952318303127, 0.99934948950195279, 0.99934797002576192, 0.99934516794789807, 0.99934253121726324,
0.99933950455047271, 0.99933859580890116, 0.99933859580890116, 0.99933843959603619, 0.99933315389179167, 0.99933295
052316362, 0.99933295052316362, 0.99933222948028877, 0.99933079133647151, 0.99932738014543765, 0.99932296628601025,
0.99932124270508171, 0.99932050496216251, 0.99931287287864 09, 0.99930973387560773, 0.99930910938047479, 0.999308483
55854973, 0.99930639572137114, 0.99930421126253199, 0.99930390952219494, 0.9993036983836544, 0.99930138579404804, 0
.99930049858553616, 0.99929669853823522, 0.99929347210236408, 0.9992908672505606, 0.99928763265363185, 0.9992844680
4421545, 0.99928379664505895, 0.99927966916826516, 0.99927773312962468, 0.99927770526082871, 0.99927709912345042, 0.
99927687784140939, 0.99927224403575499, 0.99927207381077532, 0.99927181930939668, 0.99927002405933529, 0.9992690893
8443504, 0.99926814509754935, 0.99926209779624953, 0.99926172380053946, 0.99925879119599137, 0.9992571369164428, 0.
99925543023108332, 0.99925473720415281, 0.99925405395476996, 0.99925178999752895, 0.99925125033947604, 0.9992501328
6192921, 0.99924986905882585, 0.99924920028530995, 0.99923968754348436, 0.99923459485965249, 0.99923241759069392, 0
.99923169120387256, 0.99923147007507351, 0.99923145687656278, 0.9992314203186744, 0.99923001155843205, 0.999229935 6
422919, 0.99922857165852641, 0.99922832480079427, 0.9992273500647455, 0.99922571599553478, 0.99922459451110823, 0.9
9921887338915638, 0.99921824076136967, 0.99921799594534877, 0.99921422026249507, 0.99921416096958182, 0.99921393693
880856, 0.99921214289007565, 0.99921195686836362, 0.99921173124936069, 0.99921079697872917, 0.99921079697872917, 0.
99920969500112589, 0.99920657459600304, 0.99920576086905555, 0.99920135875615668, 0.9991993742113634, 0.99919707596
521645, 0.99919679106323123, 0.9991939647148993, 0.99919331365926711, 0.9991930358487181, 0.9991923802174385, 0.999
19132428795265, 0.99919132428795265, 0.99918955771530427, 0.99918834621728014, 0.99918695370659827, 0.9991848095274
2272, 0.99918451502854677, 0.99918305007907804, 0.99918133673865761, 0.99918098513372355, 0.99918034218612761, 0.99
917569778832072, 0.99917522695365169, 0.99916961245898184, 0.99916931706102308, 0.99916622564542223, 0.999164002889
73939, 0.9991631373558626, 0.99916242662796628, 0.9991604393401492, 0.99916009288511676, 0.99915988743055761, 0.999
15624601000708, 0.99915571972563622, 0.99915453494146989, 0.9991533890068125, 0.9991533890068125, 0.999151388998771
65, 0.99915071061430993, 0.99914943422113622, 0.99914910630512976, 0.99914821757101679, 0.99914747612953936, 0.9991
4526898419131, 0.99914345069949917, 0.99914258223564578, 0.99914247592550098, 0.99914247592550098, 0.99914247592550
098, 0.99913919563374143, 0.99913892889970923, 0.99913764187338072, 0.99913725337530013, 0.99912506869755624, 0.999
12418991131657, 0.99912353284281963, 0.99912268442805241, 0.99912256080636441, 0.99912108785916554, 0.9991205087238
5605, 0.99911905183607319, 0.99911501556416027, 0.99911303344124747, 0.99911173570904621, 0.99911151344865179, 0.99
910901512122019, 0.99910901512122019, 0.99910861114278593, 0.99910858072160447, 0.99910029008761669, 0.999096598755
63084, 0.9990946029105443, 0.99909419787430709, 0.99908972705942611, 0.99908890213348012, 0.99908798052834036, 0.99
908727386792895, 0.99908547235992129, 0.999085060271661, 0.99908271261939152, 0.99908264836421112, 0.99908170787971
429, 0.99908147352610988, 0.99907734374010371, 0.99907641536622394, 0.99907617260113535, 0.99907617260113535, 0.999
07614887870355, 0.99907257552267292, 0.99907070081257165, 0.99907033448020688, 0.99906677172080538, 0.9990605827985
0363, 0.99906009578503574, 0.99905822837622016, 0.99905749278512224, 0.99905733427040067, 0.99905709346181493, 0.99
905622754399681, 0.99905530607072113, 0.99905530102905693, 0.99905508843879676, 0.99905483638272063, 0.999051498689
34995, 0.99905018462860118]
partition_check 0.313528187507
text 0.0834377172733
G.out_degree(src) 0.0628446766328
betweeness]dst 0.0599220057035
pagerank[dst] 0.0558229140606
adamic_adar 0.0553131422464
katz[src] 0.0525179131836
#_of_second_neighbors 0.0506366055407
jaccard_coefficient 0.0409049809179
closeness[src] 0.0371667200002
preferential_attachment 0.0370983238516
core_number[dst] 0.0323039599418

In [ ]:
```python
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from scipy.stats import randint as sp_randint
from time import time
# build a classifier
clf = RandomForestClassifier(n_estimators=50, n_jobs=4)
# Utility function to report best scores
def report(results, n_top=3):
    for i in range(1, n_top + 1):
        candidates = np.flatnonzero(results['rank_test_score'] == i)
        for candidate in candidates:
            print("Model with rank: {0}".format(i))
            print("Mean validation score: {0:.3f} (std: {1:.3f})".format(
                  results['mean_test_score'][candidate],
                  results['std_test_score'][candidate]))
            print("Parameters: {0}".format(results['params'][candidate]))
            print("")

# use a full grid over all parameters
param_grid = {"max_depth": range(3, 9),
              "max_features": range(2, 9),
              "min_samples_split": [2, 3, 10],
              "min_samples_leaf": [1, 3, 10],
              "bootstrap": [True, False],
              "criterion": ["gini", "entropy"]}

# run grid search
grid_search = GridSearchCV(clf, param_grid=param_grid)
start = time()
grid_search.fit(X_train_n, y_train_n)

print("GridSearchCV took %.2f seconds for %d candidate parameter settings."
      % (time() - start, len(grid_search.cv_results_['params'])))
report(grid_search.cv_results_)
with open('grid_forest.pickle', 'wb') as pfile:
        pickle.dump(grid_search.cv_results_, pfile)
```

In [ ]:
```python
import xgboost
from sklearn.model_selection import GridSearchCV
from scipy.stats import randint as sp_randint
from time import time
# build a classifier
clf = xgboost.XGBClassifier()
# Utility function to report best scores
def report(results, n_top=3):
    for i in range(1, n_top + 1):
        candidates = np.flatnonzero(results['rank_test_score'] == i)
        for candidate in candidates:
            print("Model with rank: {0}".format(i))
            print("Mean validation score: {0:.3f} (std: {1:.3f})".format(
                  results['mean_test_score'][candidate],
                  results['std_test_score'][candidate]))
            print("Parameters: {0}".format(results['params'][candidate]))
            print("")
"""(base_score=0.5, colsample_bylevel=1, colsample_bytree=1,
       gamma=0, learning_rate=0.1, max_delta_step=0, max_depth=3,
       min_child_weight=1, missing=None, n_estimators=100, nthread=-1,
       objective='binary:logistic', reg_alpha=0, reg_lambda=1,
       scale_pos_weight=1, seed=0, silent=True, subsample=1)>"""
# use a full grid over all parameters
param_grid = {
    "learning_rate": np.arange(0.15, 0.25, 0.05),
    "max_depth": range(4, 6),
    "max_delta_step": np.arange(0, 0.15, 0.05),
    "gamma": np.arange(0, 0.15, 0.05),
    "max_delta_step": np.arange(0, 0.15, 0.05),
    "subsample": np.arange(0.7, 1, 0.1),
    "min_child_weight": np.arange(0.8, 1.20, 0.1),}

# run grid search
grid_search = GridSearchCV(clf, param_grid=param_grid)
start = time()
grid_search.fit(X_train_n, y_train_n)

print("GridSearchCV took %.2f seconds for %d candidate parameter settings."
      % (time() - start, len(grid_search.cv_results_['params'])))
report(grid_search.cv_results_)
with open('grid_xgb.pickle', 'wb') as pfile:
        pickle.dump(grid_search.cv_results_, pfile)
```

In [ ]:
```
from sklearn.ensemble import VotingClassifier
xgb = xgboost.XGBClassifier()
forest = RandomForestClassifier(random_state=seed,n_estimators=80, max_features=6, max_depth=7, min_samples_leaf=3,
oob_score=True)
eclf1 = VotingClassifier(estimators=[('forest', forest), ('xgb', xgb)], voting='soft')

eclf1 = eclf1.fit(X_train_n, y_train_n)
voting_pred = eclf1.predict_proba(X_predict_n)

probs_vote = []
for t in voting_pred:
    probs_vote.append(t[1])
indices = sorted(range(len(probs_vote)), key=lambda k: probs_vote[k])[::-1][:453]
predicted_edges_vote = []
for i in range(len(indices)):
    predicted_edges_vote.append(non_edges[indices[i]])
print [probs_vote[indices[i]] for i in range(len(indices))]
with open('predicted_edges_voting.txt', 'w') as fp:
    for site_1, site_2 in predicted_edges_vote:
        fp.write(site_1+'\t'+site_2+'\n')
```

Mining data from external sources (cheating?)

Because of the public dataset we could not stand but try to obtain information from the data source (the Web!). So we downloaded/parsed the homepage of each host and obtained the links to other webpages that exist in the dataset. The search was not exhaustive(we set a strict response timeout) because it was supposed to be just a test, mostly trying to see how high such an (illegal) method would score. I believe that scoring 23% accuracy with this way proved the problems difficulty.

In [ ]:
```
import ssl
import requests
import re
from urlparse import urlparse
from bs4 import BeautifulSoup
def get_links(url):
    try:
        resp = requests.get('http://'+url,verify=False, timeout=(5,20))
    except Exception as e:
        print e
        return []
    html = resp.text
    soup = BeautifulSoup(html, "html5lib")
    links = soup.findAll("a")
    reg = '^(www.)'
    try:
        links = [re.sub(reg, '', urlparse(link.get('href')).netloc) for link in links]
    except AttributeError:
        pass
    return list(set([link for link in links if G.has_node(link) and not G.has_edge(url, link) and link !=url] ))

if os.path.isfile('cache/crawled_data.pickle'):
    with open('cache/crawled_data.pickle', 'rb') as pfile:
        text_data = pickle.load(pfile)
        print "loaded from pickle"
else:
    missing = {}
    for node in G.nodes():
        # Start the timer. Once 5 seconds are over, a SIGALRM signal is sent.
        signal.alarm(25)
        # This try/except loop ensures that
        #   you'll catch TimeoutException when it's sent.
        try:
            print node,
            missing[node]=get_links(node) # Whatever your function that might hang
        except TimeoutException:
            continue # continue the for loop if function A takes more than 5 second
        else:
            # Reset the alarm
            signal.alarm(0)
    with open('cache/crawled_data.pickle', 'wb') as pfile:
        pickle.dump(missing, pfile)

counter  = 0
for key,value in missing.iteritems():
    if len(value)>0:
        counter+=len(value)
print counter

crawled = []
for src, links in missing.iteritems():
    for dst in links:
        crawled.append((src,dst))


with open('predicted_edges_crawled.txt', 'w') as fp:
    for site_1, site_2 in crawled:
        fp.write(site_1+'\t'+site_2+'\n')
    for site_1, site_2 in predicted_edges_xgb:
        if (site_1,site_2) not in crawled:
            fp.write(site_1+'\t'+site_2+'\n')
```