Link Prediction In Social Network

Abstract

""" Given a snapshot of a social network, can we infer which new interactions among its members are likely to occur in the near future? We formalize this question as the link prediction problem, and develop approaches to link prediction based on measures for analyzing the “proximity” of nodes in a network.

"""

Introduction

""" Currently with the rapid development, online social network has been a part of people’s life. A lot of sociology, biology, and information systems can use the network to describe, in which nodes represent individual and edges represent the relationships between individuals or the interaction between individuals. Link prediction not only can be used in the field of social network but can also be applied in other fields. As in bioinformatics, link prediction can be used to discover interactions between proteins, in the field of electronic commerce, link prediction can be used to create the recommendation system and in the security field, link prediction can help to find the hidden terrorist criminal gangs . Link prediction is closely related to many areas.

"""

Problem Statement

""" Consider a social network G(V, E) at a particular time t, where V and E are sets of nodes and links, respectively. The link prediction aims to predict new links or deleted links between nodes for a future time t0(t0 > t), or missing links or unobserved links, in current network.

"""

Link Prediction Methods

I)LINK PREDICTION BY RANDOM GUESS """ Consider a social network G(V, E) no.of missing edges = |V| (|v| - 1) / 2 - |E| in sparse graph |E|<<|V|^2, probablity of correct random guess is O(1/|V|^2) """

II)LINK PREDICTION BY PROXIMITY SCORE by taking an assumption that if similiarity(proximity score) is high then chance of link between nodes is high """

Link Prediction Process

""" i) for each pair of nodes compute proximity(similiarity) score c(v1, v2)

ii) sort all pairs by the decreasing score

iii) select top n pairs(or above some threshold) as new links and recommend them """

Binary Classification

""" challenging classification problem

i)computational cost of evaluting of very large number of possible edges(quadratic in number of nodes or V^2)

ii)highly imbalanced class distribution:number of positive examples(exixting node) grow linearly and negative(missing edge) quadratically with no. of nodes (sparse) to deal with it

"0" as no edge between the nodes "1" as edge between the nodes

Exploratory Data Analysis

In [1]:

```
# Importing libraries
import networkx as nx
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import random
```

In [2]:

```
# loading csv into pandas dataframe
Train = pd.read_csv("train.csv")
```

In [3]:

```
# printing first five rows
Train.head()
```

Out[3]:

	source_node	destination_node
0	1	690569
1	1	315892
2	1	189226
3	2	834328
4	2	1615927

In [4]:

```
# we have given source and destination nodes
# how many rows and columns are there
Train.shape
```

Out[4]:

```
(9437519, 2)
```

In [5]:

```
#we have 9437519 rows and 2 columns
# Here no. of rows is actually the no of edges of graph
#no. of unique source node
len(Train.source_node.unique())
```

Out[5]:

```
1587708
```

In [6]:

```
#no of unique destination node
len(Train.destination_node.unique())
```

Out[6]:

```
1674177
```

In [7]:

```
# frequency counts for sorce_node etc.

def top_n_counts (n, col, col_1):
    gb = Train.groupby(col)[col_1].count()
    gb = gb.sort_values(ascending=False)
    return gb.head(n)

top_n_counts(15, ['source_node'], 'source_node')
```

Out[7]:

```
source_node
1492489    1566
4850        895
141489      759
735020      652
1855498     587
520731      486
631862      385
423457      375
373599      366
336432      366
591894      363
862846      351
826354      346
1368894     341
353540      341
Name: source_node, dtype: int64
```

In [8]:

```
# source node no. 1492489 have 1566 edges connected to it
# or we can say it have degree 1566
```

In [9]:

```
# constructing graph of Train Dataframe
G = nx.from_pandas_dataframe(Train ,
                             'source_node',
                             'destination_node',
                              edge_attr = True ,
                              create_using=nx.Graph()
                            )
```

In [10]:

```
# printing info of graph
print(nx.info(G))
```

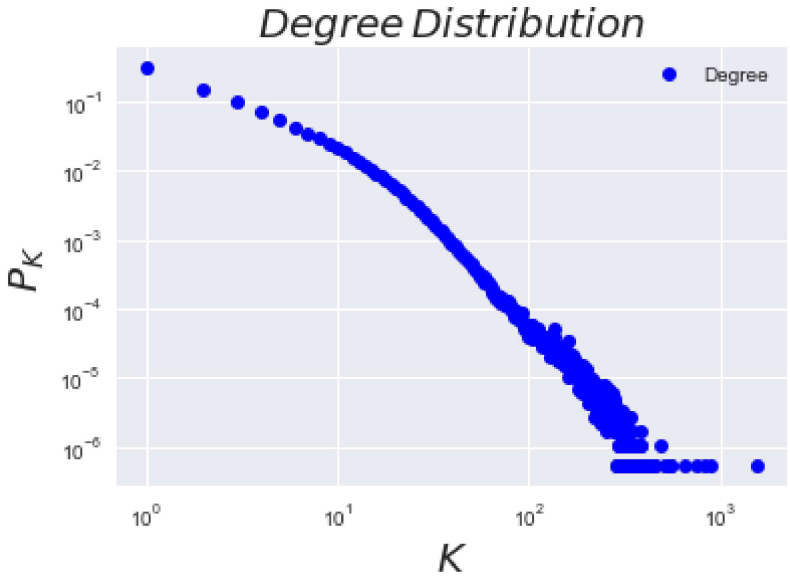
```
Name:
Type: Graph
Number of nodes: 1862220
Number of edges: 6168563
Average degree:   6.6250
```

In [11]:

```
# Plotting Degree Distribution
def plotDegreeDistribution(G):
    from collections import defaultdict
    import numpy as np
    import matplotlib.pyplot as plt
    %matplotlib inline
    degs = defaultdict(int)
    for i in G.degree().values(): degs[i]+=1
    items = sorted ( degs.items () )
    x, y = np.array(items).T
    y = [float(i) / sum(y) for i in y]
    plt.plot(x, y, 'bo')
    plt.xscale('log')
    plt.yscale('log')
    plt.legend(['Degree'])
    plt.xlabel('$K$', fontsize = 20)
    plt.ylabel('$P_K$', fontsize = 20)
    plt.title('$Degree\,Distribution$', fontsize = 20)
    plt.show()
```

In [12]:

```
plotDegreeDistribution(G)
```



In [13]:

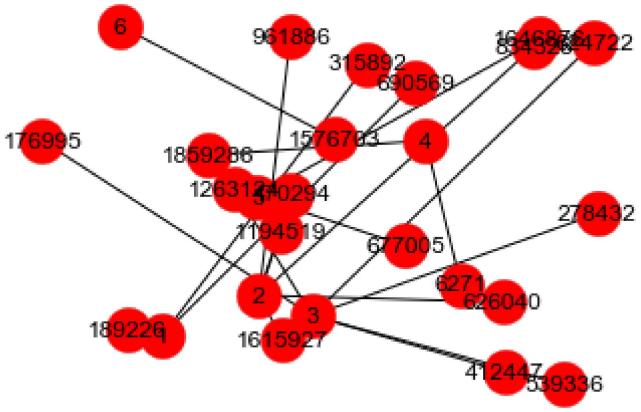
```
# Here maximum elements are having the degree from 100 to 1000
```

In [14]:

```
# how they are connected in graph
Gr = nx.from_pandas_dataframe(Train.head(20) ,
                              'source_node',
                              'destination_node',
                               edge_attr = True ,
                               create_using=nx.Graph()
                              )
```

In [15]:

```
sp =nx.random_layout(Gr)
plt.axis('off')
nx.draw_networkx(Gr, pos = sp, with_labels = True, node_size = 500,font_size=12)
plt.show()
```



In [16]:

```
# now we need more features to analyse the graph
# so first we take the no. of nodes which are in neighborhood of node
```

In [17]:

```
# Taking samples of nodes of the graph
# loading csv into pandas dataframe
Train_new = pd.read_csv("train_new.csv")
Train_new.shape
```

Out[17]:

```
(52503, 3)
```

In [18]:

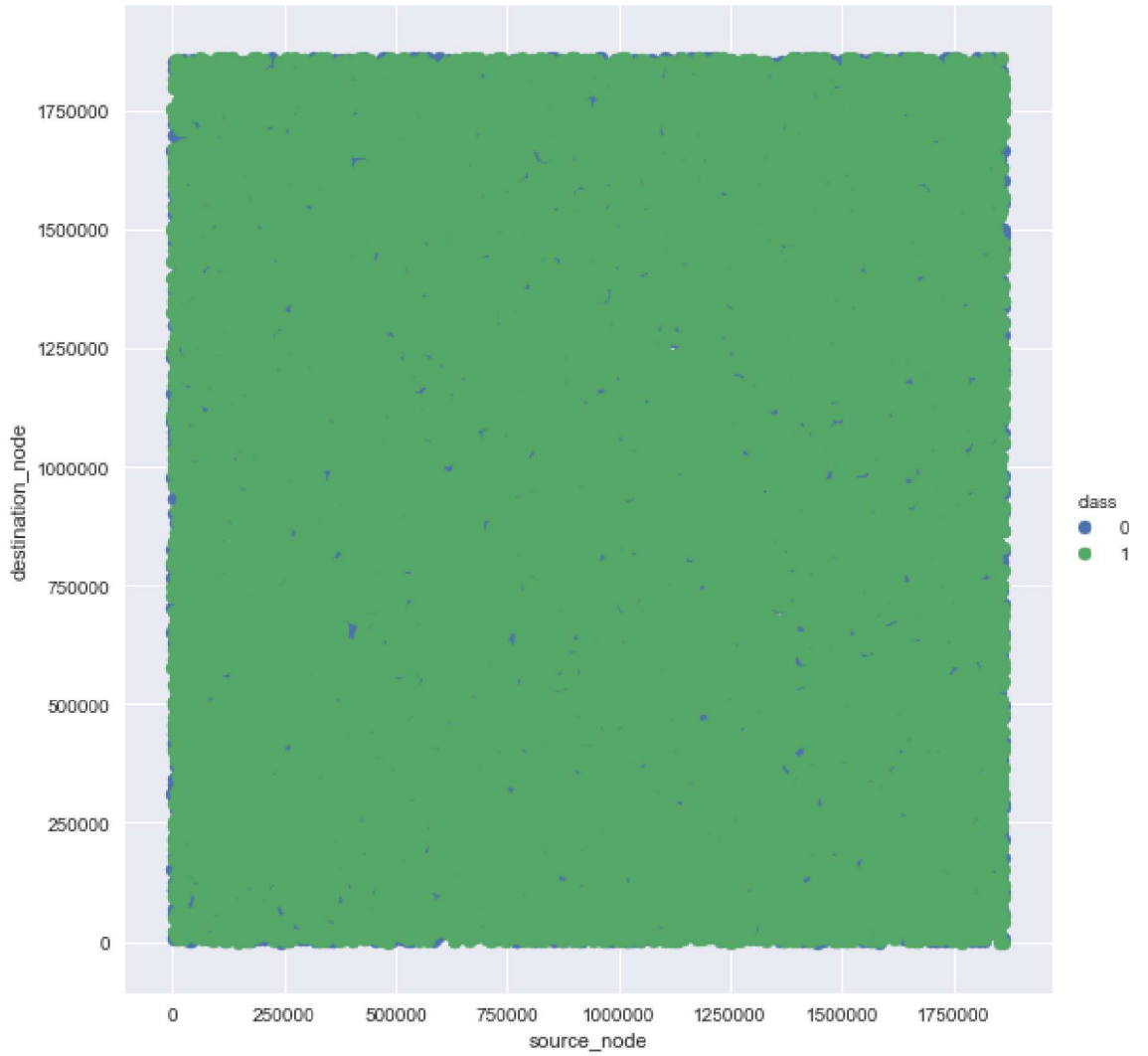
```
Train_new["class"].value_counts()
```

Out[18]:

```
1    28313
0    24190
Name: class, dtype: int64
```

In [19]:

```
sns.FacetGrid(Train_new, hue="class", size=8) \
   .map(plt.scatter, "source_node", "destination_node") \
   .add_legend();
plt.show();
```



In [20]:

```
df1 = Train_new.copy()
```

In [21]:

```
df = Train_new[['source_node','destination_node']].copy()

df2 = Train_new.copy()
df3 = Train_new.copy()
df4 = Train_new.copy()
```

Jaccard's Coefficient

The Jaccard Coefficient, also known as Jaccard index or Jaccard similarity coefficient, is a statistic measure used for comparing similarity of sample sets. It is usually denoted as J(x,y) where x and y represent two different nodes in a network. In link prediction, all the neighbours of a node are treated as a set and the prediction is done by computing and ranking the similarity of the neighbour set of each node pair. This method is based on Common Neighbours method and its complexity is also O(Nk^2). The mathematical expression of this method is as follows

```
    = |N(Vi) intersection N(Vj)| / |N(Vi) union N(Vj)|   = (common neighborhood
 / total neighborhood of Vi and Vj )

    it is normalized form of common neighbors
```

In [22]:

```
#Calculating jaccard's coefficient of nodes
grade = []
for source_node, destination_node in df.itertuples(index=False):
    preds = nx.jaccard_coefficient(G,[(source_node,destination_node)])
    for u,v,p in preds:
        grade.append(p)
df1['jaccard_coef'] = grade
```

In [23]:

```
df1.head()
```

Out[23]:

	source_node	destination_node	class	jaccard_coef
0	988311	1812513	1	0.045455
1	802218	1549012	0	0.000000
2	1334207	73229	1	0.208333
3	1772061	1443500	1	0.000000
4	282267	547056	1	0.000000

In [24]:

```
df1 = df1.reindex_axis(['source_node','destination_node','jaccard_coef','class'], axis=
1)
```

In [25]:

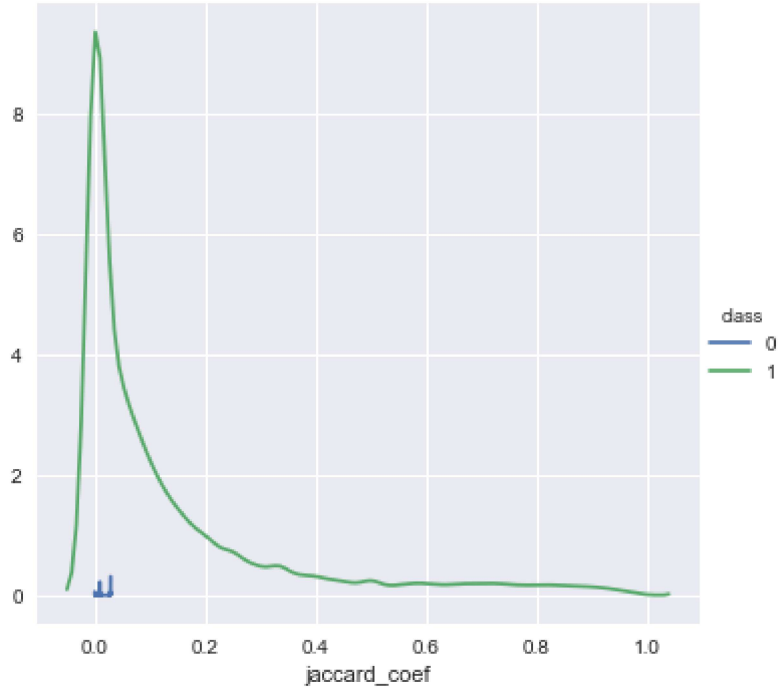
```
df1.head()
```

Out[25]:

	source_node	destination_node	jaccard_coef	class
0	988311	1812513	0.045455	1
1	802218	1549012	0.000000	0
2	1334207	73229	0.208333	1
3	1772061	1443500	0.000000	1
4	282267	547056	0.000000	1

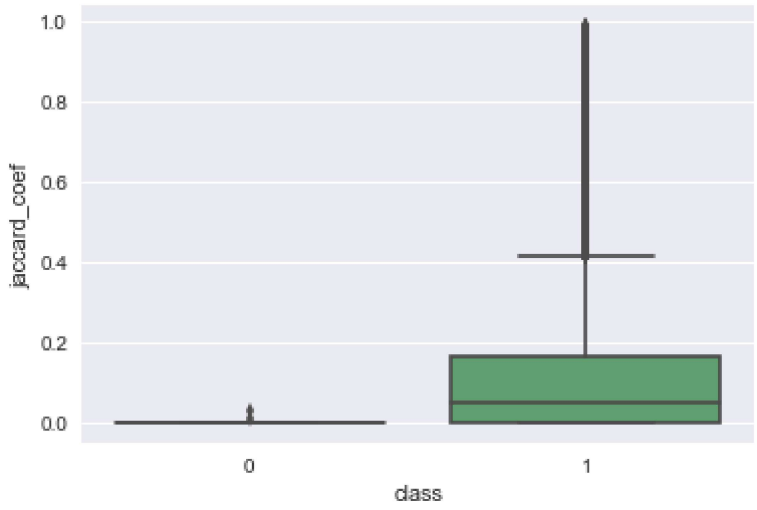
In [26]:

```
import seaborn as sns
sns.FacetGrid(df1, hue="class", size=5) \
   .map(sns.kdeplot, "jaccard_coef") \
   .add_legend();
plt.show();
```



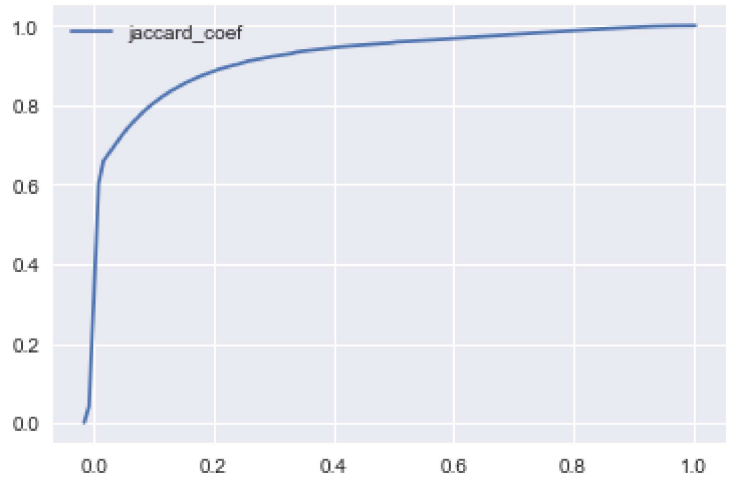
In [29]:

```
sns.boxplot(x="class",y="jaccard_coef", data=df1)
plt.show()
```



In [30]:

```
ax = sns.kdeplot(df1['jaccard_coef'], cumulative=True)
plt.show()
```



In [31]:

```
X = df1.iloc[:, 2].values  # constructing vector or matrix of independet variable
y = df1.iloc[:, 3].values # create vector of dependent variable
X = X.reshape(-1,1)
```

In [32]:

```
from sklearn.cross_validation import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 10000, random_sta
te = 0) # create four variable
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cross_validation.py:41:
DeprecationWarning: This module was deprecated in version 0.18 in favor of
the model_selection module into which all the refactored classes and funct
ions are moved. Also note that the interface of the new CV iterators are d
ifferent from that of this module. This module will be removed in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)
```

In [33]:

```
# fitting the logistic regression to  training set
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(random_state = 0,n_jobs = -1)
```

In [34]:

```
classifier.fit(X_train, y_train)
# predicting the test result
y_pred = classifier.predict(X_test)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.p
y:1228: UserWarning: 'n_jobs' > 1 does not have any effect when 'solver' i
s set to 'liblinear'. Got 'n_jobs' = -1.
  " = {}.".format(self.n_jobs))
```

In [35]:

```
# making the confusion matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
cm
```

Out[35]:

```
array([[4682,    0],
       [1971, 3347]], dtype=int64)
```

In [36]:

```
# By using jaccard's coefficient as feature we classified the data and we have 80.23% a
ccuracy
#so it is important feature for us
```

Adamic-Adar

It was initially designed to measure the relation between personal home pages. the more friends z has, the lower score it will be assigned to. Thus, the common neighbour of a pair of nodes with few neighbours contributes more to the Adamic/Adar score (AA) value than this with large number of relationships. In real-world social network, it can be interpreted as follows: if a common acquaintance of two people has more friends, then it is less likely that he will introduce the two people to each other than in the case when he has only few friends. It shows good results in predicting the friendship according to personal homepage and Wikipedia Collaboration Graph, but in the experiment of predicting author collaboration, it shows a poor accuracy prediction It is another method that is based on common neighbour; the complexity is also the O(Nk^2) . It is calculated as

Adamic-Adar index of u and v is defined as

\sum_{w \in \Gamma(u) \cap \Gamma(v)} \frac{1}{\log |\Gamma(w)|}

where \Gamma(u) denotes the set of neighbors of u.

(sum over all overlaping node and calculate 1 / log|N(V)|) # here V id degree, if degree is low then chance is high

In [37]:

```
grade = []
for source_node, destination_node in df.itertuples(index=False):
    preds = nx.adamic_adar_index(G,[(source_node,destination_node)])
    for u,v,p in preds:
        grade.append(p)
df2['adamic_adar'] = grade
```

In [38]:

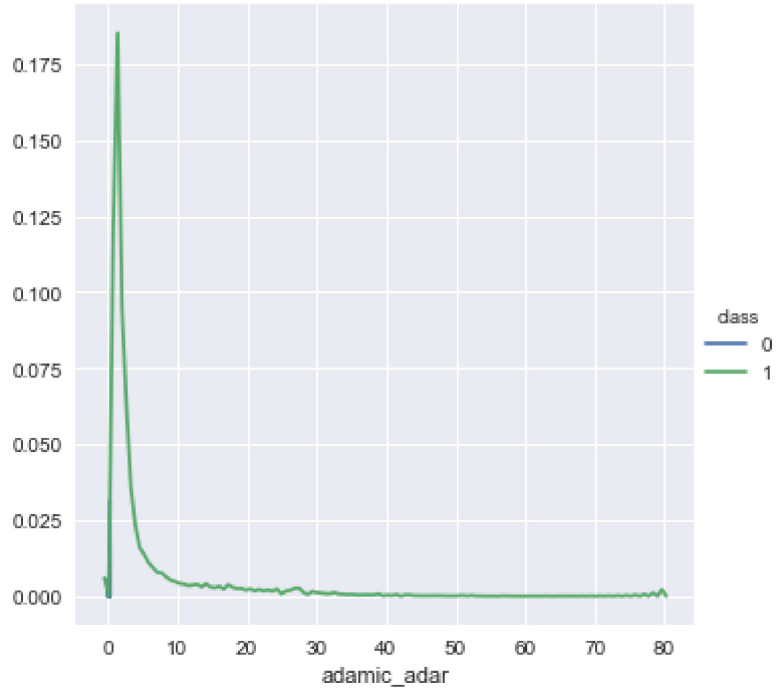
```
df2.head()
```

Out[38]:

	source_node	destination_node	class	adamic_adar
0	988311	1812513	1	0.992405
1	802218	1549012	0	0.000000
2	1334207	73229	1	2.003712
3	1772061	1443500	1	0.000000
4	282267	547056	1	0.000000

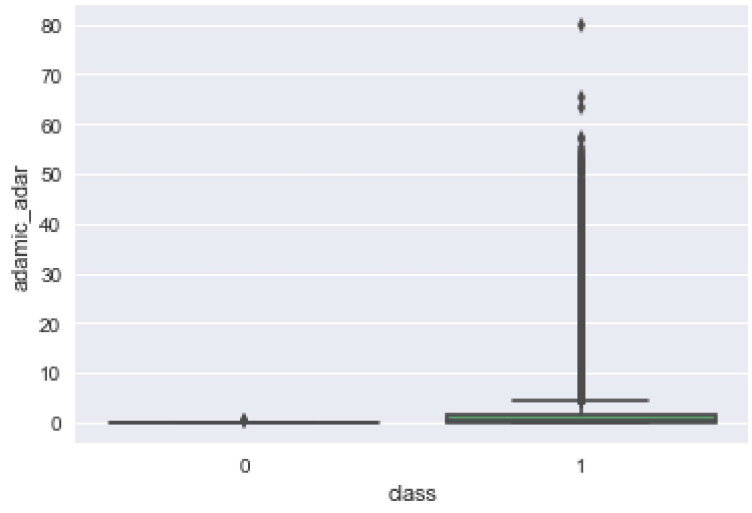
In [39]:

```
import seaborn as sns
sns.FacetGrid(df2, hue="class", size=5) \
   .map(sns.kdeplot, "adamic_adar") \
   .add_legend();
plt.show();
```



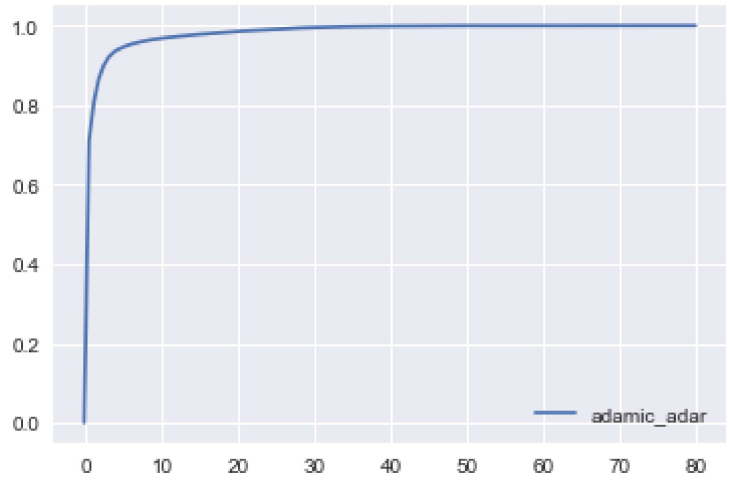
In [40]:

```
sns.boxplot(x="class",y="adamic_adar", data=df2)
plt.show()
```



In [41]:

```
ax = sns.kdeplot(df2['adamic_adar'], cumulative=True)
plt.show()
```



In [42]:

```
df2 = df2.reindex_axis(['source_node','destination_node','adamic_adar','class'], axis=1
)
df2.head()
```

Out[42]:

	source_node	destination_node	adamic_adar	class
0	988311	1812513	0.992405	1
1	802218	1549012	0.000000	0
2	1334207	73229	2.003712	1
3	1772061	1443500	0.000000	1
4	282267	547056	0.000000	1

In [43]:

```
X = df2.iloc[:, 2].values  # constructing vector or matrix of independet variable
y = df2.iloc[:, 3].values # create vector of dependent variable
X = X.reshape(-1,1)
```

In [44]:

```
from sklearn.cross_validation import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 10000, random_sta
te = 0) # create four variable
```

In [45]:

```
# fitting the logistic regression to  training set
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(random_state = 0,n_jobs = -1)
classifier.fit(X_train, y_train)
# predicting the test result
y_pred = classifier.predict(X_test)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.p
y:1228: UserWarning: 'n_jobs' > 1 does not have any effect when 'solver' i
s set to 'liblinear'. Got 'n_jobs' = -1.
  " = {}.".format(self.n_jobs))
```

In [46]:

```
# making the confusion matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
cm
```

Out[46]:

```
array([[4682,    0],
       [1855, 3463]], dtype=int64)
```

In [47]:

```
# by using adamic adar as feature we get 81.45% accuracy
```

Preferential Attachment

Due to the assumption that the node with high degree is more likely to get new links , preferential attachment was introduced as a prediction method. The degree of both nodes in a pair needs to be considered for the prediction. Same as common neighbours, this is also a basic prediction method which is usually used as a baseline to measure the performance of other prediction methods. This method will calculate similarity score for each pair of nodes within the network rather than only the neighbour of nodes; thus the complexity of preferential attachment is O(N^2k^2). This method can be expressed as

Ki and Kj are degrees of nodes, here we assign the score on the edge, by there product or sum of them Ki.Kj = |N(Vi)|.|N(Vj)| or Ki + Kj = |N(Vi)| + |N(Vj)|

In [48]:

```
grade = []
for source_node, destination_node in df.itertuples(index=False):
    preds = nx.preferential_attachment(G,[(source_node,destination_node)])
    for u,v,p in preds:
        grade.append(p)
df3['preferential_attachment']  = grade
```

In [49]:

```
df3.head()
```

Out[49]:

	source_node	destination_node	class	preferential_attachment
0	988311	1812513	1	168
1	802218	1549012	0	10
2	1334207	73229	1	168
3	1772061	1443500	1	7
4	282267	547056	1	104

In [50]:

```
df3 = df3.reindex_axis(['source_node','destination_node','preferential_attachment','cla
ss'], axis=1)
```

In [51]:

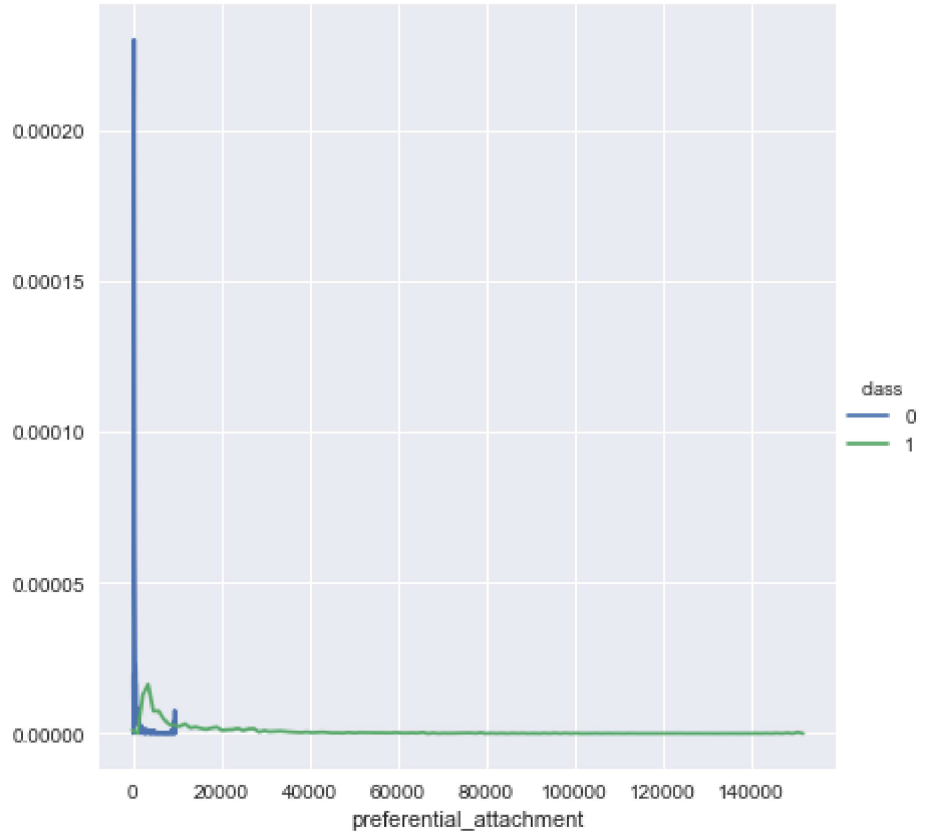
```
df3.head()
```

Out[51]:

	source_node	destination_node	preferential_attachment	class
0	988311	1812513	168	1
1	802218	1549012	10	0
2	1334207	73229	168	1
3	1772061	1443500	7	1
4	282267	547056	104	1

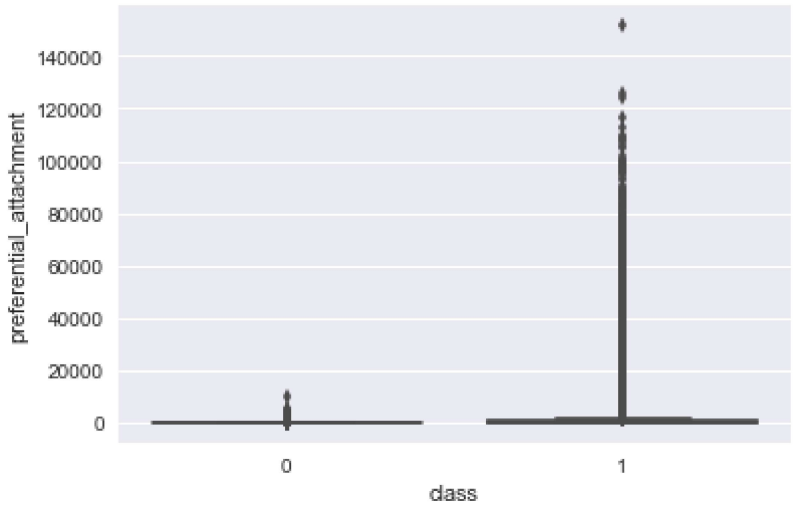
In [52]:

```
import seaborn as sns
sns.FacetGrid(df3, hue="class", size=6) \
   .map(sns.kdeplot, "preferential_attachment") \
   .add_legend();
plt.show();
```



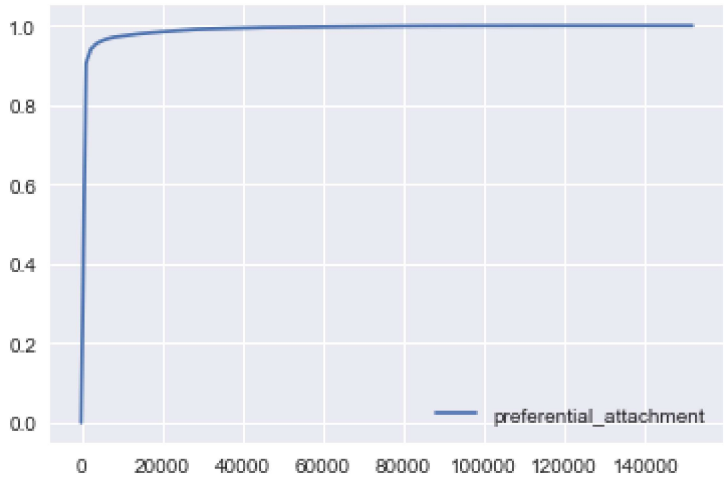
In [53]:

```
sns.boxplot(x="class",y="preferential_attachment", data=df3)
plt.show()
```



In [54]:

```
ax = sns.kdeplot(df3['preferential_attachment'], cumulative=True)
plt.show()
```



In [55]:

```
X = df3.iloc[:, 2].values  # constructing vector or matrix of independet variable
y = df3.iloc[:, 3].values # create vector of dependent variable
X = X.reshape(-1,1)
```

In [56]:

```
from sklearn.cross_validation import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 10000, random_sta
te = 0) # create four variable
```

In [57]:

```
# fitting the logistic regression to  training set
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(random_state = 0,n_jobs = -1)
classifier.fit(X_train, y_train)
# predicting the test result
y_pred = classifier.predict(X_test)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.p
y:1228: UserWarning: 'n_jobs' > 1 does not have any effect when 'solver' i
s set to 'liblinear'. Got 'n_jobs' = -1.
  " = {}.".format(self.n_jobs))
```

In [58]:

```
# making the confusion matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
cm
```

Out[58]:

```
array([[4259,  423],
       [1895, 3423]], dtype=int64)
```

In [59]:

```
#by using preferential attachment as feature we get 76.82% accuracy
```

Clustering Cofficient

For unweighted graphs, the clustering of a node u is the fraction of possible triangles through that node that exist,

c_u = \frac{2 T(u)}{deg(u)(deg(u)-1)},

where T(u) is the number of triangles through node u and deg(u) is the degree of u.

```
it measure no. of triangle
        CC(Vi).CC(Vj)

        or  CC(Vi) + CC(Vj)
```

In [60]:

```
grade = []
for source_node,destination_node in df.itertuples(index=False):

    grade.append((nx.clustering(G,source_node))*(nx.clustering(G,destination_node)))
df4['clustering_coef']  = grade
```

In [61]:

```
df4.head()
```

Out[61]:

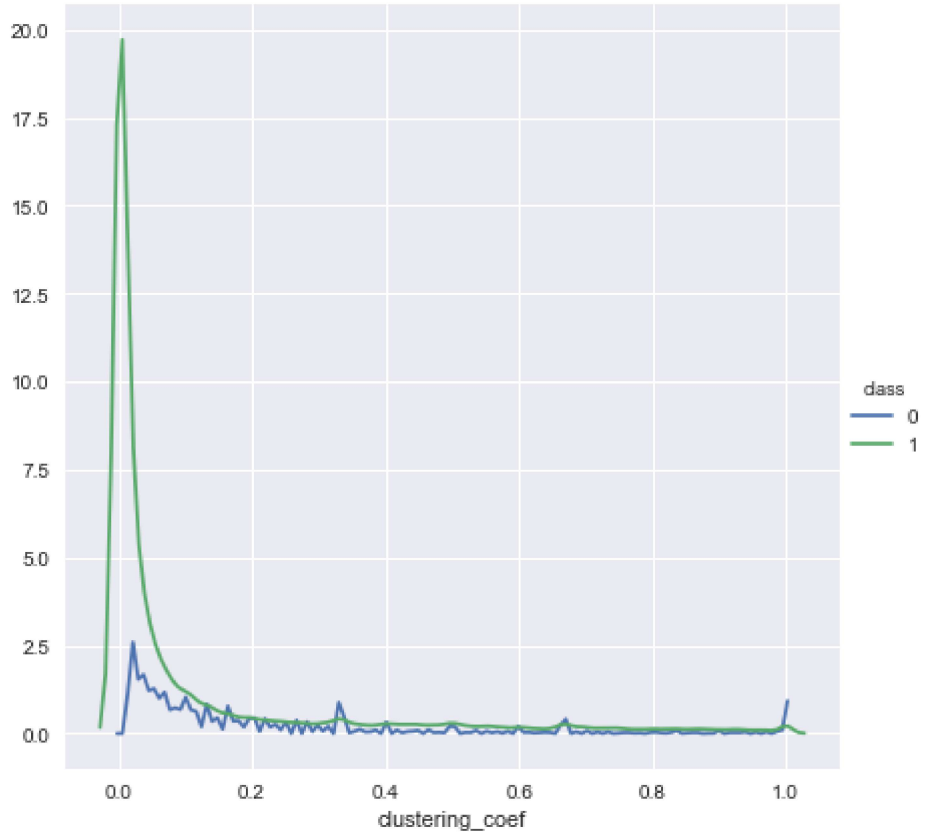
	source_node	destination_node	class	clustering_coef
0	988311	1812513	1	0.008517
1	802218	1549012	0	0.000000
2	1334207	73229	1	0.024490
3	1772061	1443500	1	0.000000
4	282267	547056	1	0.000000

In [62]:

```
df4 = df4.reindex_axis(['source_node','destination_node','clustering_coef','class'], ax
is=1)
```

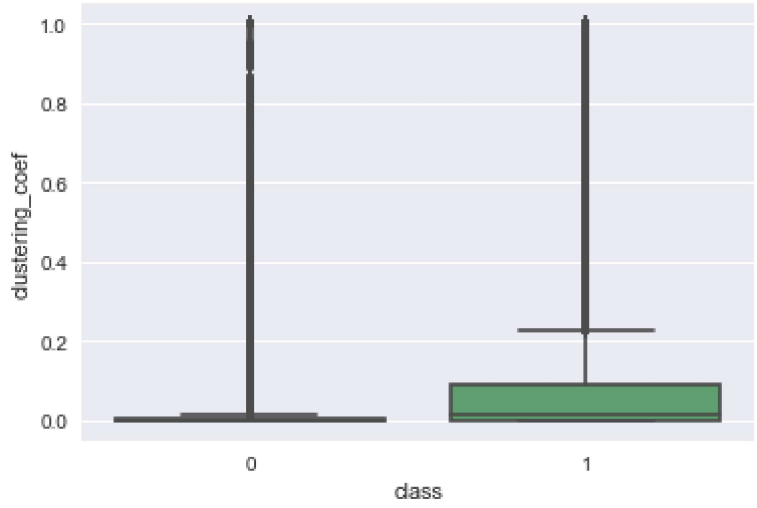
In [63]:

```
import seaborn as sns
sns.FacetGrid(df4, hue="class", size=6) \
   .map(sns.kdeplot, "clustering_coef") \
   .add_legend();
plt.show();
```



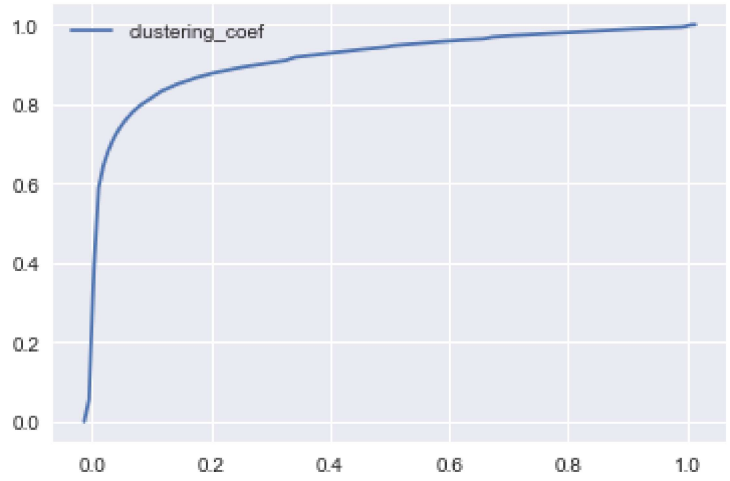
In [64]:

```
sns.boxplot(x="class",y="clustering_coef", data=df4)
plt.show()
```



In [65]:

```
ax = sns.kdeplot(df4['clustering_coef'], cumulative=True)
plt.show()
```



In [66]:

```
X = df4.iloc[:, 2].values  # constructing vector or matrix of independet variable
y = df4.iloc[:, 3].values # create vector of dependent variable
X = X.reshape(-1,1)
```

In [67]:

```
from sklearn.cross_validation import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 10000, random_sta
te = 0) # create four variable
```

In [68]:

```
# fitting the logistic regression to  training set
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(random_state = 0,n_jobs = -1)
classifier.fit(X_train, y_train)
# predicting the test result
y_pred = classifier.predict(X_test)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.p
y:1228: UserWarning: 'n_jobs' > 1 does not have any effect when 'solver' i
s set to 'liblinear'. Got 'n_jobs' = -1.
  " = {}.".format(self.n_jobs))
```

In [69]:

```
# making the confusion matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
cm
```

Out[69]:

```
array([[3444, 1238],
       [1625, 3693]], dtype=int64)
```

In [70]:

```
# by using clustering coefficient as feature we get 71.37% accuracy
```

In [71]:

```
# Combining the all features

training = Train_new.copy()
```

In [72]:

```
#Calculating jaccard's coefficient of nodes
grade = []
for source_node, destination_node in df.itertuples(index=False):
    preds = nx.jaccard_coefficient(G,[(source_node,destination_node)])
    for u,v,p in preds:
        grade.append(p)
training['jaccard_coef'] = grade
```

In [73]:

```
training['adamic_adar'] = df2['adamic_adar']
training['preferential_attachment'] = df3['preferential_attachment']
training['clustering_coef'] = df4['clustering_coef']
```

In [74]:

```
training.head()
```

Out[74]:

	source_node	destination_node	class	jaccard_coef	adamic_adar	preferential_att
0	988311	1812513	1	0.045455	0.992405	168
1	802218	1549012	0	0.000000	0.000000	10
2	1334207	73229	1	0.208333	2.003712	168
3	1772061	1443500	1	0.000000	0.000000	7
4	282267	547056	1	0.000000	0.000000	104

In [75]:

```
training = training.reindex_axis(['source_node','destination_node','jaccard_coef','adam
ic_adar','preferential_attachment','clustering_coef','class'], axis=1)
```

In [76]:

```
training.head()
```

Out[76]:

	source_node	destination_node	jaccard_coef	adamic_adar	preferential_attach
0	988311	1812513	0.045455	0.992405	168
1	802218	1549012	0.000000	0.000000	10
2	1334207	73229	0.208333	2.003712	168
3	1772061	1443500	0.000000	0.000000	7
4	282267	547056	0.000000	0.000000	104

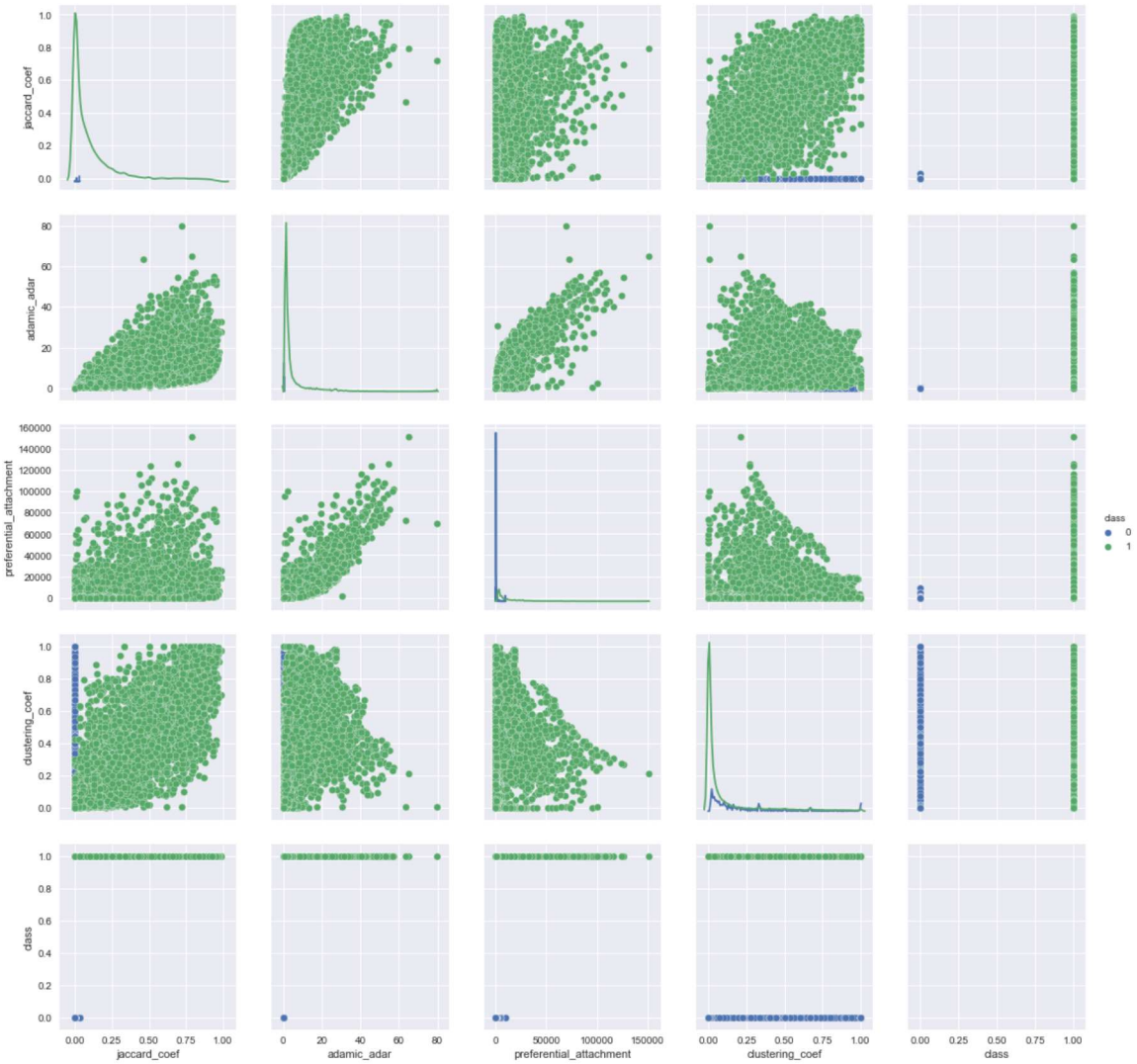
In [142]:

```
training1 = training[['jaccard_coef','adamic_adar','preferential_attachment','clusterin
g_coef','class']].copy()
```

In [144]:

```
plt.close();
sns.pairplot(training1,hue="class",size=3,diag_kind="kde");
plt.show()
```

```
C:\ProgramData\Anaconda3\lib\site-packages\statsmodels\nonparametric\kde.p
y:494: RuntimeWarning: invalid value encountered in true_divide
  binned = fast_linbin(X,a,b,gridsize)/(delta*nobs)
C:\ProgramData\Anaconda3\lib\site-packages\statsmodels\nonparametric\kdeto
ols.py:34: RuntimeWarning: invalid value encountered in double_scalars
  FAC1 = 2*(np.pi*bw/RANGE)**2
```



In [77]:

```
X = training.iloc[:, [2,3,4,5]].values  # constructing vector or matrix of independet v
ariable
y = training.iloc[:, 6].values # create vector of dependent variable
```

Train-Test Split

In [78]:

```
from sklearn.cross_validation import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 10000, random_sta
te = 0) # create four variable
```

In [107]:

```
# calculate the percentage of ones
y_test.mean()
```

Out[107]:

```
0.53180000000000005
```

In [108]:

```
# calculate the percentage of zeroes
1 - y_test.mean()
```

Out[108]:

```
0.46819999999999995
```

In [109]:

```
# calculate null accuracy(for binary classification problem coded as 0/1)
max(y_test.mean(),1 - y_test.mean())
```

Out[109]:

```
0.53180000000000005
```

Standardisation

In [79]:

```
from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)
```

Logistic Regression Classifier

In [80]:

```
# fitting the logistic regression to  training set
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(random_state = 0,n_jobs = -1)
classifier.fit(X_train, y_train)
# predicting the test result
y_pred = classifier.predict(X_test)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.p
y:1228: UserWarning: 'n_jobs' > 1 does not have any effect when 'solver' i
s set to 'liblinear'. Got 'n_jobs' = -1.
  " = {}.".format(self.n_jobs))
```

In [81]:

```
# making the confusion matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
cm
```

Out[81]:

```
array([[4608,   74],
       [1403, 3915]], dtype=int64)
```

In [82]:

```
# here we get 85.23% accuracy
```

KNN Classifier

In [83]:

```
#FITTING THE CLAASIFIER
from sklearn.neighbors import KNeighborsClassifier
classifier_knn = KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski', p = 2)
classifier_knn.fit(X_train, y_train)
y_pred = classifier_knn.predict(X_test)
```

In [84]:

```
# making the confusion matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
cm
```

Out[84]:

```
array([[4067,  615],
       [ 905, 4413]], dtype=int64)
```

In [85]:

```
# Here we get  84.80% accuracy
```

Linear SVM

In [86]:

```
from sklearn.svm import SVC
classifier_svm = SVC(kernel  = 'linear', random_state = 0)
classifier_svm.fit(X_train, y_train)
y_pred = classifier_svm.predict(X_test)
```

In [87]:

```
# making the confusion matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
cm
```

Out[87]:

```
array([[4594,   88],
       [1396, 3922]], dtype=int64)
```

In [88]:

```
#here we get 85.16% accuracy
```

RBF SVM

In [89]:

```
from sklearn.svm import SVC
classifier_rbf = SVC(kernel  = 'rbf', random_state = 0)
classifier_rbf.fit(X_train, y_train)
y_pred = classifier_rbf.predict(X_test)
```

In [90]:

```
# making the confusion matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
cm
```

Out[90]:

```
array([[4610,   72],
       [1433, 3885]], dtype=int64)
```

In [91]:

```
#here we get 84.95% accuracy
```

Gaussian Naive Bayes

In [92]:

```
from sklearn.naive_bayes import GaussianNB
classifier_nb = GaussianNB()
classifier_nb.fit(X_train, y_train)
y_pred = classifier_nb.predict(X_test)
```

In [93]:

```
# making the confusion matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
cm
```

Out[93]:

```
array([[4669,   13],
       [1694, 3624]], dtype=int64)
```

In [94]:

```
# here we get 82.93% accuracy
```

Decision Tree Classifier(using entropy and gini)

In [95]:

```
from sklearn.tree import DecisionTreeClassifier
classifier_dtc = DecisionTreeClassifier(criterion = 'entropy', random_state = 0)
classifier_dtc.fit(X_train, y_train)
y_pred = classifier_dtc.predict(X_test)
```

In [96]:

```
# making the confusion matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
cm
```

Out[96]:

```
array([[4336,  346],
       [1039, 4279]], dtype=int64)
```

In [97]:

```
# here we get 86.15% accuracy
```

In [98]:

```
# using "gini"
from sklearn.tree import DecisionTreeClassifier
classifier_dtc = DecisionTreeClassifier(criterion = 'gini', random_state = 0)
classifier_dtc.fit(X_train, y_train)
y_pred = classifier_dtc.predict(X_test)
```

In [99]:

```
# making the confusion matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
cm
```

Out[99]:

```
array([[4328,  354],
       [1047, 4271]], dtype=int64)
```

In [100]:

```
#here we gwt 85.99% accuracy
```

Random Forest Classifier

In [114]:

```
from sklearn.ensemble import RandomForestClassifier
classifier_rfc = RandomForestClassifier()
classifier_rfc.fit(X_train, y_train)
y_pred = classifier_rfc.predict(X_test)
```

In [115]:

```
# making the confusion matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
cm
```

Out[115]:

```
array([[4324,  358],
       [1007, 4311]], dtype=int64)
```

In [116]:

```
TP = cm[1,1]
TN = cm[0,0]
FP = cm[0,1]
FN = cm[1,0]
```

In [118]:

```
# Classification Accuracy,How often is the classifier  correct?
from sklearn import metrics
print((TP+TN)/float(TP+TN+FP+FN))
print(metrics.accuracy_score(y_test,y_pred))
```

```
0.8635
0.8635
```

In [119]:

```
# Classification Error,How often is the classifier incorrect ?
print((FP+FN)/float(TP+TN+FP+FN))
print(1 - metrics.accuracy_score(y_test,y_pred))
```

```
0.1365
0.1365
```

In [120]:

```
# Sensitivity:When the actual value is positive,How often is the prediction correct? or
 TPR
print((TP)/float(TP+FN))
print(metrics.recall_score(y_test,y_pred))
```

```
0.810643098909
0.810643098909
```

In [121]:

```
# Specificity: when the actual value is negative,How often is the prediction correct?
print((TN)/float(TN+FP))
```

```
0.923536950021
```

In [123]:

```
# Precision:when positive value is predicted how often is prediction correct ?
print((TP)/float(TP+FP))
print(metrics.precision_score(y_test,y_pred))
```

```
0.92332405226
0.92332405226
```

In [124]:

```
# Adjusting the classifier thresold
classifier_rfc.predict(X_test)[0:10]
```

Out[124]:

```
array([0, 0, 1, 0, 1, 1, 1, 1, 1, 0], dtype=int64)
```

In [125]:

```
# print the first 10 predicted probablities of class membership
classifier_rfc.predict_proba(X_test)[0:10, :]
```

Out[125]:

```
array([[ 0.74828886,  0.25171114],
       [ 0.7752283 ,  0.2247717 ],
       [ 0.        ,  1.        ],
       [ 0.91962069,  0.08037931],
       [ 0.3       ,  0.7       ],
       [ 0.1       ,  0.9       ],
       [ 0.        ,  1.        ],
       [ 0.        ,  1.        ],
       [ 0.        ,  1.        ],
       [ 0.95126553,  0.04873447]])
```

In [126]:

```
# print the first 10 predicted probablities of class 1
classifier_rfc.predict_proba(X_test)[0:10, 1]
```

Out[126]:

```
array([ 0.25171114,  0.2247717 ,  1.        ,  0.08037931,  0.7       ,
        0.9       ,  1.        ,  1.        ,  1.        ,  0.04873447])
```

In [128]:

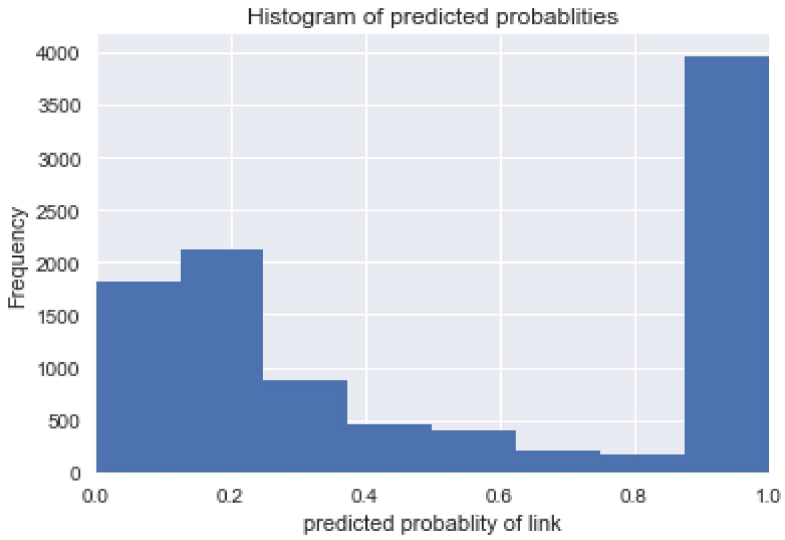
```
# store the predicted probablities for class 1
y_pred_prob = classifier_rfc.predict_proba(X_test)[:, 1]
```

In [130]:

```
%matplotlib inline
plt.rcParams['font.size']=14
#histogram of predicted probablities
plt.hist(y_pred_prob,bins = 8)
plt.xlim(0, 1)
plt.title('Histogram of predicted probablities')
plt.xlabel('predicted probablity of link')
plt.ylabel('Frequency')
```

Out[130]:

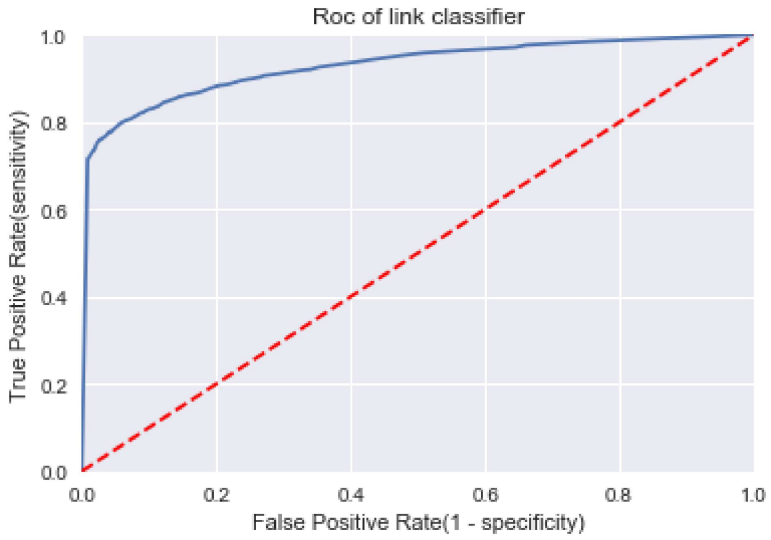
```
<matplotlib.text.Text at 0x2e8e56cf780>
```



In [134]:

```
fpr, tpr, threshold = metrics.roc_curve(y_test, y_pred_prob)
plt.title('Roc of link classifier')
plt.plot(fpr, tpr)
print(metrics.roc_auc_score(y_test, y_pred_prob))
plt.plot([0.0, 1.0], [0.0, 1.0],'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.0])
plt.ylabel('True Positive Rate(sensitivity)')
plt.xlabel('False Positive Rate(1 - specificity)')
plt.grid(True)
plt.show()
```

```
0.928287325099
```



In [137]:

```
# calculate cross validate auc
from sklearn.cross_validation import cross_val_score
cross_val_score(classifier_rfc, X, y, cv = 10,scoring ='roc_auc').mean()
```

Out[137]:

```
0.9313328276064089
```

In [141]:

```
from sklearn.metrics import classification_report
print(classification_report(y_test,y_pred))
```

```
             precision    recall  f1-score   support

          0       0.81      0.92      0.86      4682
          1       0.92      0.81      0.86      5318

avg / total       0.87      0.86      0.86     10000
```

In [103]:

```
# here we get 86.33% accuracy
```