Desempenho e Dimensionamento de Redes

Network Statistical Analysis

(Descriptors, Discriminators, Models, and Generators)

Objectives

- Statistical description of events and behaviors in networks,
- Discrimination of events and behaviors in networks,
- Mathematical modeling of network entities,
- Network activity generation.

Statistical Descriptors and Discriminators

Python references: *SciPy* – SciPy.org, http://www.scipy.org/
NumPy – NumPy Routines, https://docs.scipy.org/doc/numpy/reference/routines.html
matplotlib.pyplot - http://matplotlib.org/api/pyplot_api.html
scikit-learn, Machine Learning in Python - http://scikit-learn.org/stable/

File 'baseStats1.py' contains the base code.

It is required yours YouTube download profile (byte counts per 1 second interval), with 300 samples (5 minutes). If you did not developed a tool to obtain such profile, you may use the following *tshark* command to obtain this data:

```
    sudo tshark  -i <int_num> -z
io,stat,1,"SUM(ip.len)ip.len&&ip.src==<client_ip_address>&&ip.dst==<service_network>/24","SUM(ip
.len)ip.len&&ip.src==<service_network>/24&&ip.dst==<client_ip_address>" -q > statstable.txt
```

Identify `<int_num>` with: `sudo tshark -D`

Save the outputted statistics table in text file statstable.txt, and clean it with:

```
   cat statstable.txt | grep "<>" | head -n-1 | sed 's/[^ 0-9]//g' | sed 's/ \+/ /g'
```

Metrics

1. The file 'data1' contains the download profile data (byte counts per 1 second interval) from 40 services/users (one per column) during 5 minutes (300 samples, one per line). Import data for all datasets and plot and analyze the download profiles over time.

```
import numpy as np
import matplotlib.pyplot as plt
data1=np.loadtxt('data1')
#for dataset 0 (from 0 to 39)
plt.plot(data1[:,0],marker='s',c='blue',label='dataset 0')
plt.show()
```

2. Infer the descriptive statistical metrics of the download profiles: mean, median, variance, skewness, kurtosis, and percentiles for 1% to 100% in 1pp steps. Based on the results, identify (approximately) which profiles belong to the same class of applications.

```
import scipy.stats as stats
M=np.mean(data1,axis=0)
Md=np.median(data1,axis=0)
V=np.var(data1,axis=0)
S=stats.skew(data1)
K=stats.kurtosis(data1)
p=range(5,101,1)
Pr=np.percentile(data1,p,axis=0)
```

3. With your own YouTube download profile (measured on first assignment) identify, based only on the statistical metrics, the closest datasets to your own profile.

Probability Density Functions (PDF) and Cumulative Distribution Function (CDF)

4. Infer the empirical Probability Density Functions (PDF) and Cumulative Distribution Functions (CDF) of the download profiles for all 40 users. Analyze and comment the results. Confirm/update the mapping to classes of applications done before, now with the information about the PDFs.

```
#for dataset 0 (from 0 to 39)
pdf, bins = np.histogram(data1[:,0],bins=50,density=True)
dbin=np.diff(bins)[0]
cdf=np.cumsum(pdf)*dbin
x=bins[:-1]
plt.figure(1)
plt.plot(x,pdf,marker='s',c='blue',label='dataset 0 PDF')
plt.show()
plt.figure(2)
plt.plot(x,cdf,marker='s',c='blue',label='dataset 0 CDF')
plt.show()
```

Note: use the provided method `waitforEnter` to allow the visualization of the plots.

5. Infer the Q-Q e P-P plots for multiple pairs of download profiles and between your YouTube profile and some of the download profiles. Compare the download profiles (including your own profile), analyze the results and explain discrepancies between profiles previously mapped on the same class.

```
#Q-Q plot for dataset 0 and dataset 1
plt.figure(1)
plt.clf()
p=range(5,101,1)
Pr0=np.percentile(data1[:,0],p)
Pr1=np.percentile(data1[:,1],p)
plt.scatter(Pr0,Pr1,marker='o',c='blue')
lp=[0,max(Pr0[-1],Pr1[-1])]
plt.plot(lp,lp,c='red')
plt.show()
#P-P plot for dataset 0 and dataset 1
plt.figure(2)
plt.clf()
pdf0, bins = np.histogram(data1[:,0],bins=50,density=True)
dbin=np.diff(bins)[0]
cdf0=np.cumsum(pdf0)*dbin
#bins/xvalues of the CDF should be the same
pdf1, bins = np.histogram(data1[:,1],bins=bins,density=True)
dbin=np.diff(bins)[0]
cdf1=np.cumsum(pdf1)*dbin
plt.scatter(cdf0,cdf1,marker='o',c='blue')
lp=[min(cdf0[0],cdf1[0]),1]
plt.plot(lp,lp,c='red')
plt.show()
```

6. Calculate relative distances between PDFs using Hellinger distance. Identify the closest dataset from your YouTube profile. Confirm/update the mapping to classes of applications done before, now with the information about relative Hellinger distances.

```
def hellingerDist(pdf1, pdf2):
  return np.sqrt(np.sum((np.sqrt(pdf1) - np.sqrt(pdf2)) ** 2)) / np.sqrt(2)
```

Note: the bins/xvalues of the PDF should be the same.

7. Perform the Kolmogorov–Smirnov (KS) test for all download profiles against the exponential and normal/Gaussian distributions:

```
#for dataset 0
stats.kstest(data1[:,0],'expon')
stats.kstest(data1[:,0],'norm')
```

Perform the Kolmogorov–Smirnov test between all download profiles:

```
#for dataset 0 and dataset 1
stats.ks_2samp(data1[:,0],data1[:,1])
#for dataset 0 and dataset 30
stats.ks_2samp(data1[:,0],data1[:,30])
```

Confirm/update the mapping to classes of applications done before, now with the results from the KS tests.

Note: Both functions return D and p-value. A low p-value (less than 0.5 for a significance level of 5%) allows to reject the hypothesis that a sample follows a theoretical distribution or both samples follow the same distribution, higher values do not allow to reject the hypothesis.

Multivariate Distributions

8. The file 'data2' contains the upload profile data (byte counts per 1 second interval) for the same 40 services/users considered in file 'data1'. Infer the join PDF for download and upload profiles.

```
pdf,x,y=np.histogram2d(data1[:,0],data2[:,0],bins=10)
xx,yy = np.meshgrid(x, y)
plt.pcolormesh(xx, yy, pdf)
plt.show()
```

Aggregation Effect

9. The file 'data1All' contains the download profile data (byte counts per 1 second interval) for the 500 services/users. Aggregate the traffic from several users (first 20 to all 500), infer the PDF of the aggregate, and compared it to a Gaussian/Normal distribution with the same mean and variance. What can you conclude about the traffic profile when the aggregation level (number of users) increases?

```
for a in range(20,501,20):
  plt.clf()
  Agg=np.sum(data1All[:,0:a],axis=1)
  pdf,x=np.histogram(Agg,bins=20,density=True)
  m=np.mean(Agg)
  std=np.std(Agg)    #standard deviation = sqrt( variance )
  plt.plot(x[:-1],pdf,'k',label='empirical PDF ('+str(a)+' users)')
  plt.plot(x,mlab.normpdf(x,m,std),'r',label='inferred Gaussian PDF')
  plt.show()
  plt.legend()
  waitforEnter()
```

Events Correlation

10. The file 'traff' contains the traffic (byte counts per 1 second interval, during 5 minutes) from 10 links (one per column) within a network. It was observed a traffic anomaly (periodic bursts) on the first link (column 0), analyzing the correlation over the multiple links, identify the links over which the anomaly was propagated.

```
traff=np.loadtxt('traff')
C=abs(np.corrcoef(traff,rowvar=0))
plt.pcolormesh(C)
plt.show()
```

Periodicity

11. Infer the autocorrelation for some download profiles and identify periodic components (and periods). Compare periodic components for datasets from different classes (based on your previous mapping).

```
# for dataset 2
x=data1[:,2]
lag=np.arange(0,100,1)
xcorr=np.zeros(100)
xcorr[0]=np.correlate(x,x)
for l in lag[1:]:
     xcorr[l]=np.correlate(x[:-l],x[l:])
plt.plot(lag,xcorr)
plt.show()
```

12. Infer the periodogram for some download profiles using (i) the basic modulus-squared of the discrete FFT, and (ii) the Welch's method. Analise the outputs and identify periodic components. Compare periodic components for datasets from different classes (based on your previous mapping).

```
# for dataset 2 (with modulus-squared of FFT)
x=data1[:,2]
fft=np.fft.fft(x)
psd=abs(fft)**2
plt.plot(psd[:50])
plt.show()
# for dataset 2 (with Welch's method )
f,psd=signal.periodogram(x)
plt.plot(1/f[:50],psd[:50])
plt.show()
```

13. Analyze the module scalogram.py, import it, infer the scalogram (with a CWT using the FFT algorithm, and using the Morlet wavelet) for some download profiles and identify periodic components. Compare periodic components for datasets from different classes (based on your previous mapping).

```
import scalogram
x=data1[:,2]
scales=np.arange(1,50)
plt.ion()
plt.figure(1)
cwt=scalogram.CWTfft(x, scales)
plt.imshow(abs(cwt), cmap=plt.cm.Blues, aspect='auto')
plt.show()
plt.figure(2)
S,scales=scalogram.scalogramCWT(x,scales)
plt.plot(scales,S)
plt.show()
```

Variable Reduction

Install *scikit-learn* package: `sudo pip install sklearn`

14. Infer descriptive statistics for download and upload profiles (features): mean, variance, median, variance, skewness, kurtosis, and percentiles for 25%, 50%, 75%, 90% and 95%. Using Principal Components Analysis (PCA) reduce this variables to main components. Analysis the results and explain how to differentiate/classify services/users.

```
#features
M1=np.mean(data1,axis=0)
Md1=np.median(data1,axis=0)
V1=np.var(data1,axis=0)
S1=stats.skew(data1)
K1=stats.kurtosis(data1)
p=[25,50,75,90,95]
Pr1=np.array(np.percentile(data1,p,axis=0)).T
M2=np.mean(data2,axis=0)
Md2=np.median(data2,axis=0)
V2=np.var(data2,axis=0)
S2=stats.skew(data2)
K2=stats.kurtosis(data2)
Pr2=np.array(np.percentile(data2,p,axis=0)).T
features=np.c_[M1,M2,Md1,Md2,V1,V2,S1,S2,K1,K2,Pr1,Pr2]
#PCA
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
rcp = pca.fit(features).transform(features)
plt.scatter(rcp[0:19,0], rcp[0:19,1],marker='o',c='r',label='datasets 0-19')
plt.scatter(rcp[20:,0], rcp[20:,1],marker='s',c='b',label='datasets 20-39')
plt.legend()
plt.show()
```

Data Discrimination and Group Classification (Clustering)

15. Using the **K-means** method perform a cluster analysis with the PCA reduced data assuming 2, 3 or 4 clusters. Analyze the results and propose a traffic profile classifier.

```
from sklearn.cluster import KMeans
rcp = PCA(n_components=2).fit_transform(features)
#K-means assuming 2 clusters
kmeans = KMeans(init='k-means++', n_clusters=2)
kmeans.fit(rcp)
kmeans.labels_
#vizualization plot
x_min, x_max = 1.5*rcp[:, 0].min(), 1.5*rcp[:, 0].max()
y_min, y_max = 1.5*rcp[:, 1].min(), 1.5*rcp[:, 1].max()
N=20
hx=(x_max-x_min)/N
hy=(y_max-y_min)/N
xx, yy = np.meshgrid(np.arange(x_min, x_max, hx), np.arange(y_min, y_max, hy))
Z = kmeans.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
plt.figure(7)
plt.imshow(Z, interpolation='nearest',extent=(xx.min(), xx.max(),
     yy.min(), yy.max()),cmap=plt.cm.Blues,aspect='auto', origin='lower',alpha=0.7)
plt.plot(rcp[:, 0], rcp[:, 1], 'ko')
# Plot the centroids as a white X
centroids = kmeans.cluster_centers_
plt.scatter(centroids[:, 0], centroids[:, 1],marker='x', color='r')
plt.xlim(xx.min(), xx.max())
plt.ylim(yy.min(), yy.max())
plt.xticks(())
plt.yticks(())
plt.show()
```

16. Using the **DBSCAN** method perform a cluster analysis with the PCA reduced data assuming neighborhood maximum distance (eps) of 1e8, 1e10 and 1e11. Analyze the results and propose a traffic profile classifier.

```
        from sklearn.cluster import DBSCAN
        rcp = PCA(n_components=2).fit_transform(features)
        #DBSCAN assuming a neighborhood maximum distance of 1e11
        dbscan = DBSCAN(eps=1e11)
        dbscan.fit(rcp)
        L=dbscan.labels_
        print(L)
        colors = plt.cm.Blues(np.linspace(0, 1, len(set(L))))
        plt.figure(1)
        for l in set(L):
            p=(L==l)
            if l==-1:
                color='r'
            else:
                color=colors[l]
            plt.plot(rcp[p,0],rcp[p,1],'o',c=color,markersize=10)
        plt.show()
```

Note: a sample with label "-1" represents a noise/anomaly value, i.e., a not classifiable value.

Anomaly Identification

17. Assuming that data (PCA reduced profile data) has 20% of anomalies, infer the anomaly boundaries using the elliptic envelope method.

```
        from sklearn.covariance import EllipticEnvelope
        anom_perc=20
        clf=EllipticEnvelope(contamination=.1)
        clf.fit(rcp)
        clf.decision_function(rcp).ravel()
        pred=clf.decision_function(rcp).ravel()
        threshold = stats.scoreatpercentile(pred,anom_perc)
        Anom=pred>threshold
        print(Anom)
        Z = clf.decision_function(np.c_[xx.ravel(), yy.ravel()])
        Z = Z.reshape(xx.shape)
        plt.contourf(xx, yy, Z, levels=np.linspace(Z.min(), threshold, 7),cmap=plt.cm.Blues_r)
        plt.contour(xx, yy, Z, levels=[threshold],linewidths=2, colors='red')
        plt.plot(rcp[:, 0], rcp[:, 1], 'ko')
        plt.show()
```

Mathematical Models and Generators

Single Distribution Models

18. Choose a download profile from 'data1' file. Model data based on Exponential, Gaussian, and empirical discrete distribution. Generate synthetic traffic profiles, plot profiles over time and compare the resulting PDF.

```
x=data1[:,2]
M=np.mean(x)
V=np.var(x)
H,bins=np.histogram(x,bins=20)
probs=1.0*H/np.sum(H)
#Exponential
ye=np.random.exponential(M,(300,20))
#Gaussian/Normal
yg=np.random.normal(M,V**0.5,(300,20))
#Empirical discrete
yd=np.random.choice(bins[:-1],(300,20),p=probs)
```

Machine State Modulated Distributions

19. Choose a download profile from 'data1' file. Identify activity (ON) and non-activity (OFF) periods. Model the duration of permanence in each state by an exponential distribution, and the activity rate by a discrete distribution. Generate synthetic traffic profiles, plot profiles over time and compare the resulting PDF and main descriptive statistics.

Trend/Growth Models

20. Based on the monthly traffic volume data (in TBytes, 1x10^{12} bytes) from the Amsterdam IX (json data file 'ams-ix-traffic.json') since 2001, infer the growth behavior and rate, assuming linear and exponential rates. Predict when the input traffic volume will exceed 1.5EBytes (Exa Bytes = 1x10^{18} bytes).

```
import json
from scipy.optimize import curve_fit
with open('ams-ix-traffic.json') as data_file:
    data=json.loads(data_file)
Xout=[]
for monthT in data:
    Xout.append(monthT['out_traffic'])

def linearG(t,Y0,R):
    return Y0+R*t

def expG(t,Y0,A0,R):
    return Y0+A0*np.exp(R*t)

t=np.arange(0,len(Xout))
paramsL, cov = curve_fit(linearG,t,Xout)
paramsE, cov = curve_fit(expG,t,Xout,[500,1,.01])

plt.plot(t,Xout,'k')
plt.plot(t,linearG(t,paramsL[0],paramsL[1]),'b')
plt.plot(t,expG(t,paramsE[0],paramsE[1],paramsE[2]),'r')
plt.show()
```