In [1]:

```
import pandas as pd
from sklearn import preprocessing
```

In [2]:

```
from keras.models import Sequential
from keras.layers import Dense
from keras.utils import to_categorical
from keras.callbacks import EarlyStopping
```

```
Using TensorFlow backend.
```

In [3]:

```
import matplotlib.pyplot as plt
import seaborn as sns
```

In [4]:

```
from sklearn.decomposition import PCA
```

In [5]:

```
train_df = pd.read_csv("wat-time-interval-10000.csv")
```

In [6]:

```
train_df.head()
```

Out[6]:

	time_intervals	r0	r1	r2	r3	r4	r5	r6	r7	r8	...	inport_east	inport_west	o
0	10000	613	194	105	48	385	110	105	70	264	...	179	200	
1	10000	675	825	201	95	445	521	179	117	300	...	351	393	
2	20000	707	860	384	171	419	406	182	76	207	...	453	548	
3	20000	629	266	202	89	337	122	129	50	175	...	232	327	
4	30000	672	266	172	86	316	108	80	65	200	...	293	362	

5 rows × 34 columns

In [7]:

```
train_X = train_df.drop(columns=['time_intervals','target'])
```

In [8]:

```
train_X
```

Out[8]:

	r0	r1	r2	r3	r4	r5	r6	r7	r8	r9	...	inport_south	inport_east	inport_w
0	613	194	105	48	385	110	105	70	264	153	...	711	179	2
1	675	825	201	95	445	521	179	117	300	423	...	1076	351	3
2	707	860	384	171	419	406	182	76	207	264	...	1085	453	5
3	629	266	202	89	337	122	129	50	175	86	...	700	232	3
4	672	266	172	86	316	108	80	65	200	78	...	586	293	3
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
625	559	229	168	102	329	90	96	72	239	59	...	417	498	4
626	486	282	228	120	192	66	102	84	109	55	...	459	312	4
627	503	269	198	114	210	72	90	96	126	66	...	450	324	4
628	747	369	288	198	366	134	114	90	204	98	...	618	496	6
629	762	348	258	180	390	144	102	90	227	107	...	585	510	5

630 rows × 32 columns

In [9]:

```
x = train_X.values #returns a numpy array
min_max_scaler = preprocessing.MinMaxScaler()
x_scaled = min_max_scaler.fit_transform(x)
train_X = pd.DataFrame(x_scaled)
```

In [10]:

```
corr_df = pd.concat([train_X, train_df[['target']]], axis = 1)
corr_df.corr()
```

Out[10]:

	0	1	2	3	4	5	6	7	
0	1.000000	0.880315	0.921483	0.843099	0.967894	0.779053	0.893604	0.843027	0.9
1	0.880315	1.000000	0.928026	0.874358	0.838893	0.902357	0.820267	0.800133	0.8
2	0.921483	0.928026	1.000000	0.914807	0.862259	0.758288	0.834735	0.780864	0.8
3	0.843099	0.874358	0.914807	1.000000	0.772269	0.739808	0.683599	0.837076	0.7
4	0.967894	0.838893	0.862259	0.772269	1.000000	0.819686	0.936351	0.847388	0.9
5	0.779053	0.902357	0.758288	0.739808	0.819686	1.000000	0.807417	0.811944	0.7
6	0.893604	0.820267	0.834735	0.683599	0.936351	0.807417	1.000000	0.777446	0.8
7	0.843027	0.800133	0.780864	0.837076	0.847388	0.811944	0.777446	1.000000	0.8
8	0.948662	0.806011	0.838658	0.754123	0.979600	0.765015	0.885505	0.831592	1.0
9	0.827457	0.886674	0.790391	0.764939	0.859911	0.908614	0.782777	0.820263	0.8
10	0.860886	0.787954	0.820909	0.698767	0.892640	0.695586	0.876315	0.743136	0.9
11	0.796771	0.763521	0.756715	0.814951	0.788356	0.696688	0.635050	0.863269	0.8
12	0.908541	0.776521	0.806334	0.721701	0.924290	0.718594	0.833637	0.770290	0.9
13	0.822733	0.843763	0.764902	0.738184	0.838230	0.855968	0.768893	0.797249	0.8
14	0.787450	0.791578	0.759947	0.658551	0.809367	0.751011	0.819728	0.711299	0.8
15	0.746831	0.745869	0.718016	0.763804	0.726475	0.691399	0.634277	0.828203	0.7
16	0.928516	0.944195	0.898233	0.833638	0.937455	0.912105	0.893230	0.849538	0.9
17	0.929048	0.944010	0.898550	0.833667	0.938177	0.911528	0.893659	0.848722	0.9
18	0.970968	0.875951	0.924178	0.870777	0.943584	0.772211	0.873211	0.876091	0.9
19	0.971068	0.875429	0.924264	0.870371	0.943806	0.770952	0.873267	0.874444	0.9
20	0.970680	0.956778	0.936481	0.853645	0.957091	0.883817	0.916250	0.848737	0.9
21	0.920258	0.901315	0.855046	0.781116	0.957884	0.901952	0.905633	0.849926	0.9
22	0.955375	0.949886	0.950114	0.911695	0.927164	0.862797	0.869727	0.878804	0.9
23	0.951722	0.851538	0.870375	0.803435	0.968424	0.807444	0.897488	0.872173	0.9
24	0.968819	0.916082	0.960831	0.930476	0.919094	0.791908	0.850631	0.890207	0.9
25	0.970715	0.956777	0.936627	0.853811	0.957128	0.883655	0.916264	0.848743	0.9
26	0.951749	0.851558	0.870393	0.803472	0.968449	0.807479	0.897516	0.872176	0.9
27	0.968830	0.916106	0.960823	0.930422	0.919087	0.791940	0.850628	0.890231	0.9
28	0.920094	0.901163	0.854758	0.780908	0.957746	0.901946	0.905419	0.849886	0.9
29	0.955349	0.949885	0.950103	0.911616	0.927126	0.862834	0.869793	0.878714	0.9
30	0.974987	0.938291	0.935989	0.872597	0.967581	0.871561	0.910754	0.882269	0.9
31	0.974987	0.938291	0.935989	0.872597	0.967581	0.871561	0.910754	0.882269	0.9
target	-0.005309	-0.053992	-0.020834	-0.018737	-0.007652	-0.061772	-0.019972	-0.014347	-0.0

33 rows × 33 columns

In [11]:

```
train_Y = train_df['target']
```

In [12]:

```
train_Y
```

Out[12]:

```
0      1
1      0
2      0
3      1
4      1
      ..
625    1
626    0
627    1
628    1
629    0
Name: target, Length: 630, dtype: int64
```

In [13]:

```
model = Sequential()
```

In [14]:

```
n_cols = train_X.shape[1]
n_cols
```

Out[14]:

```
32
```

In [15]:

```
model.add(Dense(16, activation='relu', input_shape=(n_cols,)))
model.add(Dense(8, activation='relu'))
model.add(Dense(4, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

In [16]:

```
model.compile(optimizer='sgd', loss='mean_squared_error', metrics=['accuracy'])
```

In [17]:

```
early_stopping_monitor = EarlyStopping(patience=5)
```

In [18]:

```
model.fit(train_X, train_Y, epochs=11, validation_split=0.9, callbacks=[early_stopping_mon
itor])
```

```
Train on 62 samples, validate on 568 samples
Epoch 1/11
62/62 [==============================] - 1s 9ms/step - loss: 0.2582 - accurac
y: 0.5000 - val_loss: 0.2500 - val_accuracy: 0.4877
Epoch 2/11
62/62 [==============================] - 0s 414us/step - loss: 0.2578 - accur
acy: 0.4677 - val_loss: 0.2499 - val_accuracy: 0.4877
Epoch 3/11
62/62 [==============================] - 0s 458us/step - loss: 0.2575 - accur
acy: 0.4839 - val_loss: 0.2499 - val_accuracy: 0.4912
Epoch 4/11
62/62 [==============================] - 0s 457us/step - loss: 0.2571 - accur
acy: 0.4839 - val_loss: 0.2499 - val_accuracy: 0.4965
Epoch 5/11
62/62 [==============================] - 0s 500us/step - loss: 0.2567 - accur
acy: 0.4516 - val_loss: 0.2499 - val_accuracy: 0.5035
Epoch 6/11
62/62 [==============================] - 0s 437us/step - loss: 0.2564 - accur
acy: 0.4516 - val_loss: 0.2498 - val_accuracy: 0.5106
Epoch 7/11
62/62 [==============================] - 0s 390us/step - loss: 0.2562 - accur
acy: 0.5000 - val_loss: 0.2498 - val_accuracy: 0.5070
Epoch 8/11
62/62 [==============================] - 0s 461us/step - loss: 0.2558 - accur
acy: 0.5000 - val_loss: 0.2498 - val_accuracy: 0.5000
Epoch 9/11
62/62 [==============================] - 0s 411us/step - loss: 0.2556 - accur
acy: 0.5161 - val_loss: 0.2498 - val_accuracy: 0.5000
Epoch 10/11
62/62 [==============================] - 0s 358us/step - loss: 0.2552 - accur
acy: 0.5000 - val_loss: 0.2498 - val_accuracy: 0.5088
Epoch 11/11
62/62 [==============================] - 0s 341us/step - loss: 0.2550 - accur
acy: 0.5000 - val_loss: 0.2498 - val_accuracy: 0.5018
```

Out[18]:

```
<keras.callbacks.callbacks.History at 0x1c2a35c9688>
```

In [19]:

```
pred = model.predict(train_X)
```

In [20]:

```
for i in range(100):
    print("%s, %s" % (pred[i], train_Y[i]))
```

```
[0.5312084], 1
[0.60701454], 0
[0.5745477], 0
[0.5392393], 1
[0.54078215], 1
[0.5917756], 0
[0.5424941], 1
[0.5731483], 0
[0.5078099], 1
[0.5334772], 0
[0.5321795], 0
[0.52892166], 1
[0.57510674], 0
[0.5209706], 1
[0.47567695], 0
[0.47758237], 1
[0.5098469], 0
[0.52366406], 1
[0.49425402], 0
[0.4930328], 1
[0.4909404], 0
[0.49737042], 1
[0.5200422], 0
[0.5154869], 1
[0.5023351], 1
[0.48712125], 0
[0.48930284], 1
[0.4914095], 0
[0.48737836], 0
[0.50134146], 1
[0.5007277], 0
[0.50170285], 1
[0.49653998], 1
[0.5013529], 0
[0.50661314], 1
[0.5006046], 0
[0.5011329], 0
[0.50091493], 1
[0.49561423], 0
[0.49567387], 1
[0.4965696], 1
[0.5012371], 0
[0.49946064], 0
[0.49579704], 1
[0.5006046], 1
[0.49760163], 0
[0.50135124], 1
[0.51587296], 0
[0.5114503], 1
[0.5081869], 0
[0.50972253], 0
[0.5054434], 1
[0.4965696], 0
[0.50526625], 1
[0.50273174], 0
[0.49520078], 1
[0.50070244], 0
```

```
[0.50170285], 1
[0.49653998], 1
[0.5001148], 0
[0.50661314], 1
[0.5009712], 0
[0.5026751], 1
[0.4996968], 0
[0.506493], 0
[0.49473664], 1
[0.49415794], 0
[0.51657057], 1
[0.5201761], 1
[0.50170285], 0
[0.49628782], 1
[0.490821], 0
[0.5066395], 0
[0.5011149], 1
[0.506493], 1
[0.5023422], 0
[0.5067134], 1
[0.50972253], 0
[0.4965696], 0
[0.5023892], 1
[0.50273174], 0
[0.49561423], 1
[0.50070244], 0
[0.50645095], 1
[0.5023422], 1
[0.5001148], 0
[0.49881086], 1
[0.5009712], 0
[0.50124395], 0
[0.4959606], 1
[0.506493], 0
[0.5020691], 1
[0.4939502], 0
[0.4972177], 1
[0.50074697], 0
[0.4994048], 1
[0.49628782], 1
[0.49049947], 0
[0.5066395], 0
[0.5011149], 1
```

In [21]:

```
pca = PCA(n_components = 2)
```

In [22]:

```
pca.fit(train_X)
```

Out[22]:

```
PCA(copy=True, iterated_power='auto', n_components=2, random_state=None,
    svd_solver='auto', tol=0.0, whiten=False)
```

In [23]:

```
principal_components = pca.transform(train_X)
principal_components
```

Out[23]:

```
array([[ 1.42274132, -0.48488112],
       [ 3.04674493, -0.44801   ],
       [ 2.97389447, -0.05414932],
       ...,
       [ 1.21177165,  0.18454936],
       [ 2.06891045,  0.16933166],
       [ 2.02289304,  0.01123124]])
```

In [24]:

```
pca.explained_variance_ratio_
```

Out[24]:

```
array([0.89532433, 0.02839152])
```

In [25]:

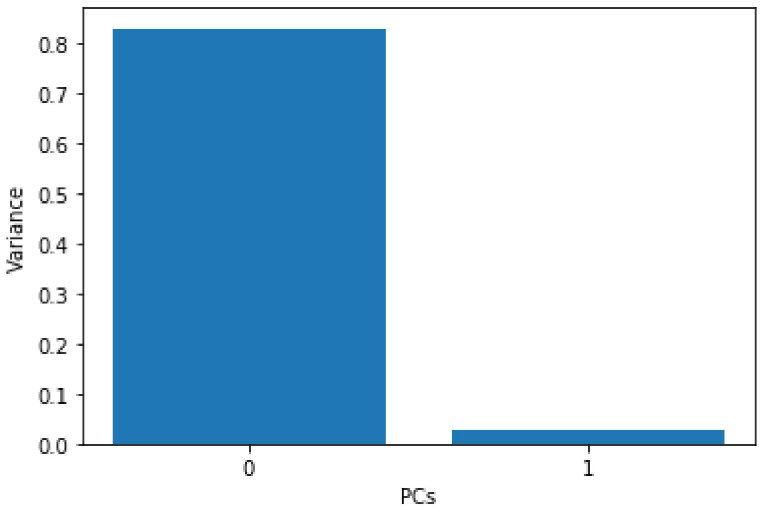
```
features = range(pca.n_components_)
```

In [26]:

```
plt.bar(features, pca.explained_variance_)
plt.xticks(features)
plt.xlabel("PCs")
plt.ylabel("Variance")
```

Out[26]:

```
Text(0, 0.5, 'Variance')
```



In [27]:

```
principal_df = pd.DataFrame(data = principal_components , columns = ['pc 1', 'pc 2'])
principal_df
```

Out[27]:

	pc 1	pc 2
0	1.422741	-0.484881
1	3.046745	-0.448010
2	2.973894	-0.054149
3	1.444013	-0.256094
4	1.539798	-0.223460
...	...	...
625	1.626957	-0.278583
626	1.194475	0.205759
627	1.211772	0.184549
628	2.068910	0.169332
629	2.022893	0.011231

630 rows × 2 columns

In [28]:

```
final_df = pd.concat([principal_df, train_df[['target']]], axis = 1)
final_df
```

Out[28]:

	pc 1	pc 2	target
0	1.422741	-0.484881	1
1	3.046745	-0.448010	0
2	2.973894	-0.054149	0
3	1.444013	-0.256094	1
4	1.539798	-0.223460	1
...	...	...	...
625	1.626957	-0.278583	1
626	1.194475	0.205759	0
627	1.211772	0.184549	1
628	2.068910	0.169332	1
629	2.022893	0.011231	0

630 rows × 3 columns

In [29]:

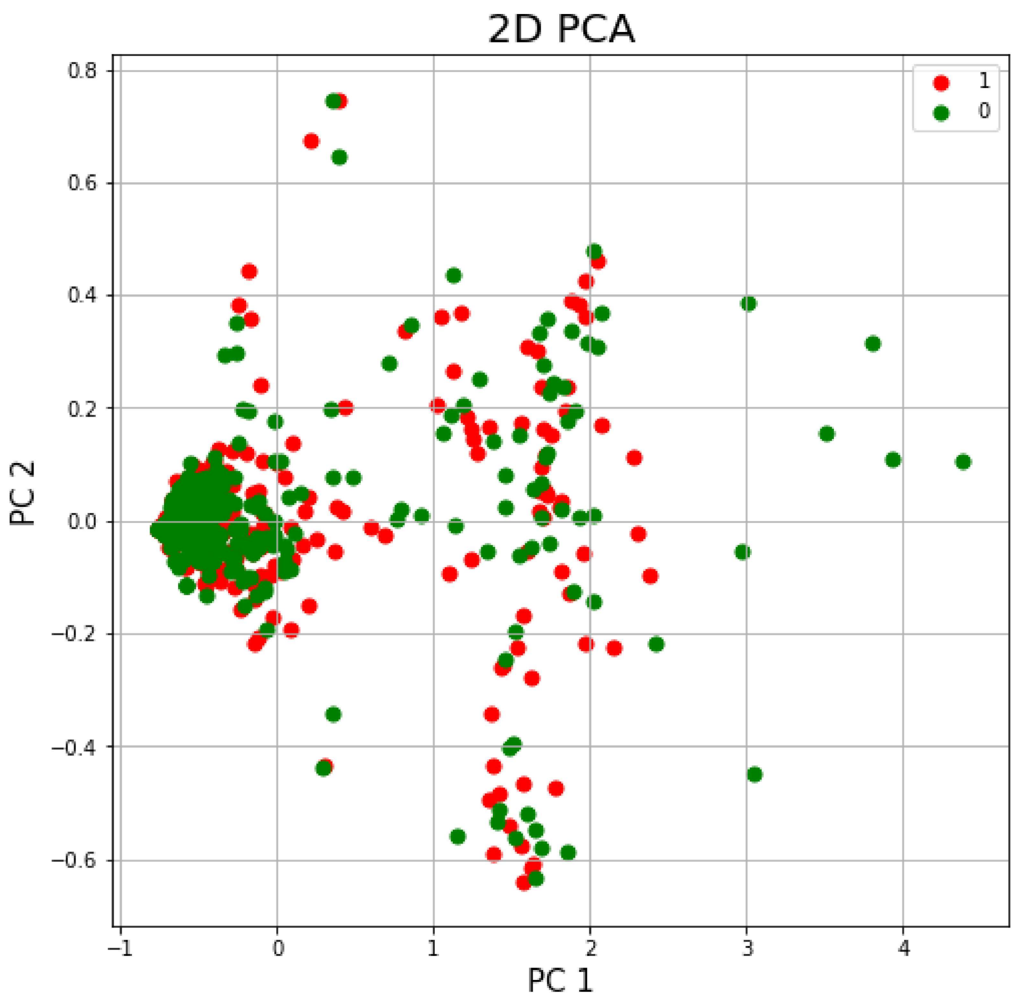
```
final_df.corr()
```

Out[29]:

	pc 1	pc 2	target
pc 1	1.000000e+00	4.802458e-15	-0.016670
pc 2	4.802458e-15	1.000000e+00	-0.016496
target	-1.667040e-02	-1.649636e-02	1.000000

In [30]:

```
fig = plt.figure(figsize = (8,8))
ax = fig.add_subplot(1,1,1)
ax.set_xlabel('PC 1', fontsize = 15)
ax.set_ylabel('PC 2', fontsize = 15)
ax.set_title('2D PCA', fontsize = 20)
targets = [1, 0]
colors = ['r', 'g']
for target, color in zip(targets,colors):
    indicesToKeep = final_df['target'] == target
    ax.scatter(final_df.loc[indicesToKeep, 'pc 1']
               , final_df.loc[indicesToKeep, 'pc 2']
               , c = color
               , s = 50)
ax.legend(targets)
ax.grid()
```

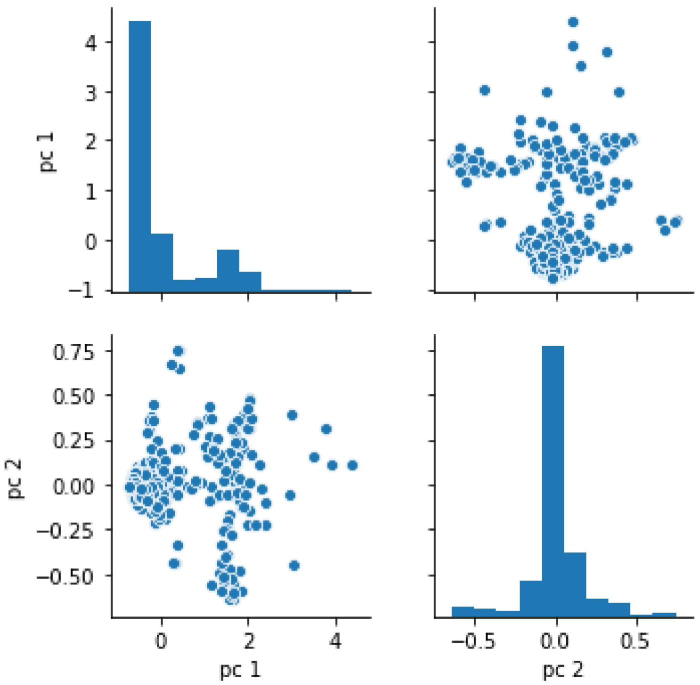


In [31]:

```
sns.pairplot(final_df.loc[:,final_df.dtypes == 'float64'])
```

Out[31]:

<seaborn.axisgrid.PairGrid at 0x1c2a6d5c648>



In [32]:

```
corr_df
```

Out[32]:

	0	1	2	3	4	5	6	7	8
0	0.720000	0.179298	0.197368	0.186047	0.753425	0.211132	0.475113	0.312500	0.880000
1	0.792941	0.762477	0.377820	0.368217	0.870841	1.000000	0.809955	0.522321	1.000000
2	0.830588	0.794824	0.721805	0.662791	0.819961	0.779271	0.823529	0.339286	0.690000
3	0.738824	0.245841	0.379699	0.344961	0.659491	0.234165	0.583710	0.223214	0.583333
4	0.789412	0.245841	0.323308	0.333333	0.618395	0.207294	0.361991	0.290179	0.666667
...	...	...	...	...	...	...	...	...	...
625	0.656471	0.211645	0.315789	0.395349	0.643836	0.172745	0.434389	0.321429	0.796667
626	0.570588	0.260628	0.428571	0.465116	0.375734	0.126679	0.461538	0.375000	0.363333
627	0.590588	0.248614	0.372180	0.441860	0.410959	0.138196	0.407240	0.428571	0.420000
628	0.877647	0.341035	0.541353	0.767442	0.716243	0.257198	0.515837	0.401786	0.680000
629	0.895294	0.321627	0.484962	0.697674	0.763209	0.276392	0.461538	0.401786	0.756667

630 rows × 33 columns

In [33]:

```
corr_df[corr_df.duplicated()].shape
```

Out[33]:

(71, 33)

In [34]:

```
dup_df = train_df.drop(columns=['time_intervals'])
dup_df
```

Out[34]:

	r0	r1	r2	r3	r4	r5	r6	r7	r8	r9	...	inport_east	inport_west	outport_lo
0	613	194	105	48	385	110	105	70	264	153	...	179	200	
1	675	825	201	95	445	521	179	117	300	423	...	351	393	1
2	707	860	384	171	419	406	182	76	207	264	...	453	548	1
3	629	266	202	89	337	122	129	50	175	86	...	232	327	
4	672	266	172	86	316	108	80	65	200	78	...	293	362	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
625	559	229	168	102	329	90	96	72	239	59	...	498	403	
626	486	282	228	120	192	66	102	84	109	55	...	312	474	
627	503	269	198	114	210	72	90	96	126	66	...	324	449	
628	747	369	288	198	366	134	114	90	204	98	...	496	621	
629	762	348	258	180	390	144	102	90	227	107	...	510	570	

630 rows × 33 columns

In [35]:

```
dup_df[dup_df.duplicated()].shape
```

Out[35]:

(71, 33)

In [36]:

```
dup_df[dup_df.duplicated()].count()
```

Out[36]:

```
r0               71
r1               71
r2               71
r3               71
r4               71
r5               71
r6               71
r7               71
r8               71
r9               71
r10              71
r11              71
r12              71
r13              71
r14              71
r15              71
pkt_get          71
pkt_data         71
pkt_put          71
pkt_ack          71
inport_local     71
inport_north     71
inport_south     71
inport_east      71
inport_west      71
outport_local    71
outport_north    71
outport_south    71
outport_east     71
outport_west     71
target           71
tot_packets      71
tot_mean         71
dtype: int64
```

In [37]:

```
print ((71/630)*100)
```

```
11.26984126984127
```