InputSensitivityRanking

January 29, 2019

PFE
Adds
–
Imports
–

In [1]:
```
import os
import pandas
import numpy as np


NB_CONFIGURATIONS = 1052

## We are studying two quantitative/performance properties:
# elapsed time
# and size of the output
# we have two distinct datasets for both

predDimension = "elapsedtime" # "size" #

### It is simple/convenient to fix the "order" of dataset once and for all
# (listing can be sensitive to eg an operating system)
### if you want you can use this method
#dataFolder="./datay4m"
#listeAdresse = []
#adresseIni = os.listdir(dataFolder)
#for video in adresseIni:
#    listeRep = os.listdir(dataFolder + "/" + video)
#    for rep in listeRep:
#        listeAdresse.append(dataFolder + "/" + video + "/" + rep)

# dataFolder="./datacalda"
# all processing using the same exact cluster on IGRIDA (calda) and video format (y4m)
# (experiments suggest that hardware or video format does influence execution time)

dataTimeFolder = './datacalda2/'
dataSizeFolder = './datay4m2/'
```

```python
def mkDataTime():
    return [dataTimeFolder + 'x264-1908-bridgefar-wasm/x264-results1.csv',
 dataTimeFolder + 'x264-1908-ice-wasm/x264-results1.csv',
 dataTimeFolder + 'x264-1908-flower-wasm/x264-results1.csv',
# './datacalda/x264-0408-tos3k-wasm/x264-results1.csv', # can't retrieve the original vi
 dataTimeFolder + 'x264-1908-caire-wasm/x264-results1.csv',
# './datacalda/x264-0308-sintel-wasm/x264-results1.csv', # same as calda for time
 dataTimeFolder + 'x264-0208-sintel-calda-wasm/x264-results1.csv', # representative vide
 dataTimeFolder + 'x264-1908-footballcif-wasm/x264-results1.csv',
 dataTimeFolder + 'x264-0308-crowd_run-wasm/x264-results1.csv',
 dataTimeFolder + 'x264-0608-blue-wasm/x264-results1.csv',
 dataTimeFolder + 'x264-0608-people-wasm/x264-results1.csv',
 dataTimeFolder + 'x264-1908-sunflowers-wasm/x264-results1.csv',
 dataTimeFolder + 'x264-0408-deadline-wasm/x264-results1.csv',
 dataTimeFolder + 'x264-2108-bridgeclose-wasm/x264-results1.csv',
 dataTimeFolder + 'x264-1908-husky-wasm/x264-results1.csv',
 dataTimeFolder + 'x264-1908-tennis-wasm/x264-results1.csv',
 dataTimeFolder + 'x264-1908-riverbed-wasm/x264-results1.csv',
 dataTimeFolder + 'x264-0608-park-wasm/x264-results1.csv',
 dataTimeFolder + 'x264-0508-soccer-wasm/x264-results1.csv']

# dataFolder="./datay4m"
# all processing using the same video format (y4m)
# (experiments confirm that hardware/cluster does not change anything about the size)
def mkDataSize():
    return [dataSizeFolder + 'x264-1908-akiyo-wasm/x264-results1.csv',
 dataSizeFolder + 'x264-1908-bridgefar-wasm/x264-results1.csv',
 dataSizeFolder + 'x264-1908-football15-wasm/x264-results1.csv',
 dataSizeFolder + 'x264-0608-tractor-wasm/x264-results1.csv',
 dataSizeFolder + 'x264-1908-ice-wasm/x264-results1.csv',
 dataSizeFolder + 'x264-1908-students-wasm/x264-results1.csv',
 dataSizeFolder + 'x264-1908-flower-wasm/x264-results1.csv',
# './datay4m/x264-0408-tos3k-wasm/x264-results1.csv', # can't retrieve the original vide
 dataSizeFolder + 'x264-1908-caire-wasm/x264-results1.csv',
# './datay4m/x264-0308-sintel-wasm/x264-results1.csv', # same as calda for size
 dataSizeFolder + 'x264-0208-sintel-calda-wasm/x264-results1.csv', # representative vide
 dataSizeFolder + 'x264-0308-ducks-wasm/x264-results1.csv',
 dataSizeFolder + 'x264-1908-footballcif-wasm/x264-results1.csv',
 dataSizeFolder + 'x264-0308-crowd_run-wasm/x264-results1.csv',
 dataSizeFolder + 'x264-0608-blue-wasm/x264-results1.csv',
 dataSizeFolder + 'x264-0608-people-wasm/x264-results1.csv',
 dataSizeFolder + 'x264-1908-sunflowers-wasm/x264-results1.csv',
 dataSizeFolder + 'x264-2108-netflix-wasm/x264-results1.csv',
 dataSizeFolder + 'x264-0408-deadline-wasm/x264-results1.csv',
# './datay4m/x264-0208-crowd_run-bermuda-wasm/x264-results1.csv', # same as crowd above
 dataSizeFolder + 'x264-2108-bridgeclose-wasm/x264-results1.csv',
 dataSizeFolder + 'x264-1908-husky-wasm/x264-results1.csv',
```

```
 dataSizeFolder + 'x264-1908-waterfall-wasm/x264-results1.csv',
 dataSizeFolder + 'x264-0308-mobilesif-wasm/x264-results1.csv',
 dataSizeFolder + 'x264-1908-tennis-wasm/x264-results1.csv',
 # './datay4m/x264-0408-football-wasm/x264-results1.csv', # same as football15
 dataSizeFolder + 'x264-1908-riverbed-wasm/x264-results1.csv',
 dataSizeFolder + 'x264-0608-park-wasm/x264-results1.csv',
 dataSizeFolder + 'x264-0508-soccer-wasm/x264-results1.csv']

# the key idea is to have in the *same order* video (for the size dataset or the time da
# ie each "common" video will have the same "identifier"
def mkData():

    sizeAlignment = [x.replace(dataSizeFolder, '') for x in mkDataSize()]
    timeAlignment = [x.replace(dataTimeFolder, '') for x in mkDataTime()]
    common = []
    for s in sizeAlignment:
        if s in timeAlignment:
            common.append(s)

    common = np.sort(common)

    specificSize = []
    for s in sizeAlignment:
        if s not in timeAlignment:
            specificSize.append(s)

    specificSize = np.sort(specificSize)

    # unnecessary
    specificTime = []
    for t in timeAlignment:
        if t not in sizeAlignment:
            specificTime.append(t)

    # time datas are subsets of size datas
    assert(len(specificSize) + len(common) == len(sizeAlignment))
    assert (len(specificTime) == 0)

    if predDimension == "size":     # mkDataSize()
        return list(map(lambda s: dataSizeFolder + s, np.append(common, specificSize)))
    elif predDimension == "elapsedtime":     #mkDataTime()
        return list(map(lambda s: dataTimeFolder + s, np.append(common, specificTime)))
    else:
        print("Error (pred dimension unknown)")

listeAdresse = mkData() # mkDataTime() #
if predDimension == "size":
    assert(len(listeAdresse) == len(mkDataSize()))
```

```
elif predDimension == "elapsedtime":
    assert(len(listeAdresse) == len(mkDataTime()))
#print(np.sort(mkDataSize()))
#print(np.sort(mkDataTime()))

#print(np.intersect1d(mkDataSize(), mkDataTime()))



# creation of the list of videos (for each video: x264 configurations + measurements)
listeVideo = []
for adresse in listeAdresse:
    listeVideo.append(pandas.read_csv(open(adresse,"r")))
# test
print("There are " + str(len(listeVideo)) + " videos")
assert (len(listeAdresse) == len(listeVideo))
listeAdresse
#vidEx = listeVideo[0][0:5]
#vidEx.drop(['usertime', 'systemtime'], axis=1)
#pd.DataFrame(listeVideo[2])#['elapsedtime']
#listeVideo[1]
```

```
There are 17 videos
```

```
Out[1]: ['./datacalda2/x264-0208-sintel-calda-wasm/x264-results1.csv',
         './datacalda2/x264-0308-crowd_run-wasm/x264-results1.csv',
         './datacalda2/x264-0408-deadline-wasm/x264-results1.csv',
         './datacalda2/x264-0508-soccer-wasm/x264-results1.csv',
         './datacalda2/x264-0608-blue-wasm/x264-results1.csv',
         './datacalda2/x264-0608-park-wasm/x264-results1.csv',
         './datacalda2/x264-0608-people-wasm/x264-results1.csv',
         './datacalda2/x264-1908-bridgefar-wasm/x264-results1.csv',
         './datacalda2/x264-1908-caire-wasm/x264-results1.csv',
         './datacalda2/x264-1908-flower-wasm/x264-results1.csv',
         './datacalda2/x264-1908-footballcif-wasm/x264-results1.csv',
         './datacalda2/x264-1908-husky-wasm/x264-results1.csv',
         './datacalda2/x264-1908-ice-wasm/x264-results1.csv',
         './datacalda2/x264-1908-riverbed-wasm/x264-results1.csv',
         './datacalda2/x264-1908-sunflowers-wasm/x264-results1.csv',
         './datacalda2/x264-1908-tennis-wasm/x264-results1.csv',
         './datacalda2/x264-2108-bridgeclose-wasm/x264-results1.csv']
```

Configurations sorting

–

```
In [2]: dico = {}
        for i in listeVideo:
            for j in range(len(i)):
```

```
                if i["configurationID"][j] not in dico.keys():
                    dico[i["configurationID"][j]]=i[predDimension][j]
                else :
                    dico[i["configurationID"][j]]=dico[i["configurationID"][j]]+i[predDimension]
```

```
In [3]: dico2 = {}
        for i in listeVideo:
            for j in range(len(i)):
                if i["configurationID"][j] not in dico2.keys():
                    dico2[i["configurationID"][j]]=[i[predDimension][j]]
                else :
                    dico2[i["configurationID"][j]].append(i[predDimension][j])
```

```
In [4]: res = pandas.DataFrame.from_dict(dico, orient='index')
        res.reset_index(inplace= True)
        res.columns=['configid','sum']
        res.sort_values("sum",inplace=True)
        print(res[0:2])
        print("...")
        print(res[1150:1152])
```

```
     configid        sum
880       754  37.570505
715       605  37.588692
...
     configid        sum
372       297  94.651103
627       526  94.871240
```

We add all the time of all inputs, and calculate the sum of it by config before sorting. We can see that the difference between the first and the last configurations (*2.5 in time)

```
In [5]: res2 = pandas.DataFrame.from_dict(dico2, orient='index')
        res2.sum(axis = 1)
        res3 = res2.transpose()
        res3.describe().transpose()[0:5]
        # res3.describe().transpose().sort_values(by="mean")
```

Out[5]:

	count	mean	std	min	25%	50%	75%	max
1	17.0	3.952804	4.050540	0.3954	0.7326	3.074800	5.010375	13.631667
10	17.0	5.169551	5.379713	0.5016	0.9908	4.299091	5.902625	17.598667
100	17.0	2.237163	2.113802	0.2342	0.4228	2.039800	2.765500	7.135333
1000	17.0	3.782660	3.796787	0.3250	0.6040	3.353800	4.699750	12.297000
1001	17.0	3.392039	3.278258	0.3256	0.6888	2.970800	4.329125	11.056000

Correlations matrix about Kullback-Leiber divergence

–

In [6]:
```python
import scipy.stats as sc
import numpy as np
from scipy import stats
import matplotlib.pyplot as plt
from scipy.cluster.hierarchy import dendrogram, linkage

taille = len(listeVideo)

divKLTaille = [[0 for x in range(taille)] for y in range(taille)]
divKLTaille2 = [[0 for x in range(taille)] for y in range(taille)]

for i in range(taille):
    for j in range(taille):
        divKLTaille[i][j] = sc.entropy(pk=listeVideo[i]['size'],
                                       qk=listeVideo[j]['size'])

indiceTaille = dendrogram(linkage(divKLTaille, 'ward'), no_plot=True)['leaves']

for i in range(taille):
    for j in range(taille):
        divKLTaille2[i][j] = sc.entropy(pk=listeVideo[indiceTaille[i]]['size'],
                                        qk=listeVideo[indiceTaille[j]]['size'])

plt.subplots(figsize=(10, 10))
plt.imshow(divKLTaille2,cmap='Reds',interpolation='nearest')
plt.title('div_KL of size')
plt.xticks(range(len(indiceTaille)),indiceTaille)
plt.yticks(range(len(indiceTaille)), indiceTaille)
plt.colorbar()
plt.show()

divKLTemps = [[0 for x in range(taille)] for y in range(taille)]
divKLTemps2 = [[0 for x in range(taille)] for y in range(taille)]

for i in range(taille):
    for j in range(taille):
        divKLTemps[i][j] = sc.entropy(pk=listeVideo[i][predDimension],
                                      qk=listeVideo[j][predDimension])

indiceTemps = dendrogram(linkage(divKLTemps, 'ward'), no_plot=True)['leaves']

for i in range(taille):
    for j in range(taille):
        divKLTemps2[i][j] = sc.entropy(pk=listeVideo[indiceTemps[i]][predDimension],
                                       qk=listeVideo[indiceTemps[j]][predDimension])


plt.subplots(figsize=(10, 10))
```

```
    plt.imshow(divKLTemps2,cmap='Reds',interpolation='nearest')
    plt.title('div_KL of time')
    plt.xticks(range(len(indiceTemps)),indiceTemps)
    plt.yticks(range(len(indiceTemps)), indiceTemps)
    plt.colorbar()
    plt.show()
```

```
<Figure size 1000x1000 with 2 Axes>
```

```
<Figure size 1000x1000 with 2 Axes>
```

We need one mean to compare all the clustering we have done. What differency them? General function for transfering video i on video j

–

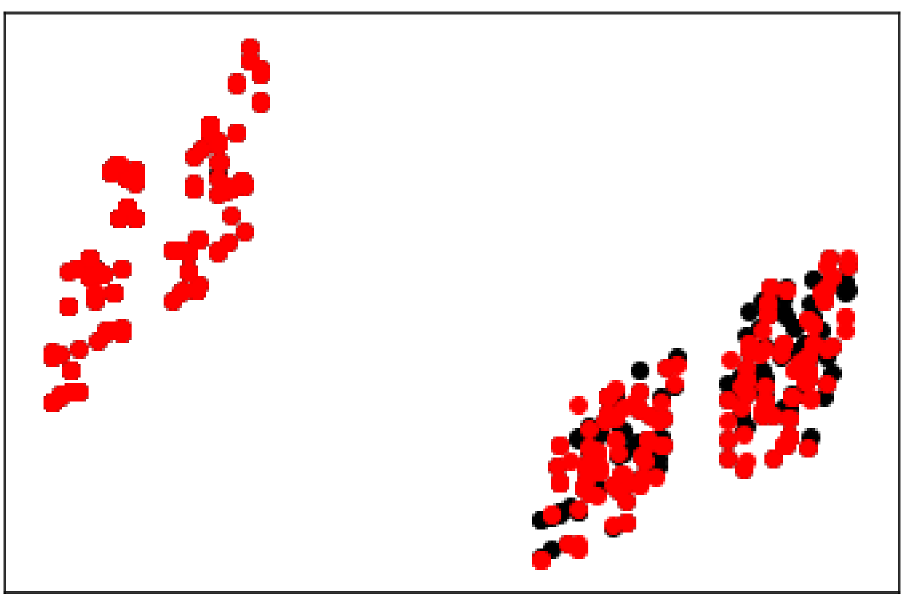
```
In [7]: import matplotlib.pyplot as plt
        import numpy as np
        from sklearn.model_selection import train_test_split
        from sklearn.metrics import mean_squared_error, r2_score
        from sklearn.svm import SVC, SVR
        from sklearn import datasets, linear_model
        from sklearn.ensemble import RandomForestRegressor

        def transfer(var,varexp1,varexp2,i,j,testSize,method):
            # where var is either 'size' or 'elapsedtime'
            # varexp1 & varexp2 two parameter of configuration
            # i is the number of the "learning" video
            # j is the video which will benefits from the learning of i
            # testSize is the size of the test dataset (70 for 70% of tests)
            # method is 'sv' for support vector, 'rf' for random forest, 'reg' for regression

            st = testSize/100

            # Split the targets into training/testing sets
            x_train, x_test, y_train, y_test = train_test_split(listeVideo[i][[var, varexp1,vare
                                                                listeVideo[j][var],
                                                                test_size= st,
                                                                random_state=0)

            #choose the method
            if method == 'reg':
                clf = linear_model.LinearRegression()

            if method == 'rf':
                clf = RandomForestRegressor(n_estimators=20)

            if method == 'sv':
```

```
    clf = SVC(kernel='rbf', C=1e10, gamma=1e-8)

# Apply the model to the training datasets and predict for the testing dataset
y_pred = clf.fit(x_train, y_train).predict(x_test)

# Explained variance score: 1 is perfect prediction
print('Variance score: %.2f' % r2_score(y_test, y_pred))

# Then we plot the prediction vs the reality
plt.scatter(x_test['size'], y_test,  color='black')
plt.scatter(x_test['size'], y_pred,  color='red')
plt.xticks(())
plt.yticks(())
plt.show()
```

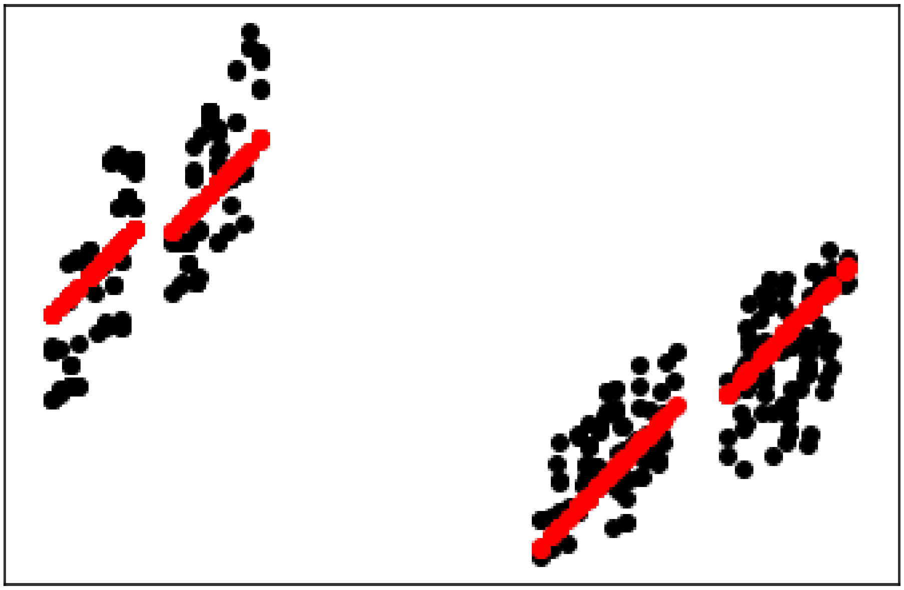
We test the function on tranfer with video 1 and video 5

```
In [8]: print("svm")
        transfer('size','no_mbtree','no_cabac',1,6,30,'sv')
        print("reg")
        transfer('size','no_mbtree','no_cabac',1,6,30,'reg')
        print("random forest")
        transfer('size','no_mbtree','no_cabac',1,6,30,'rf')
```

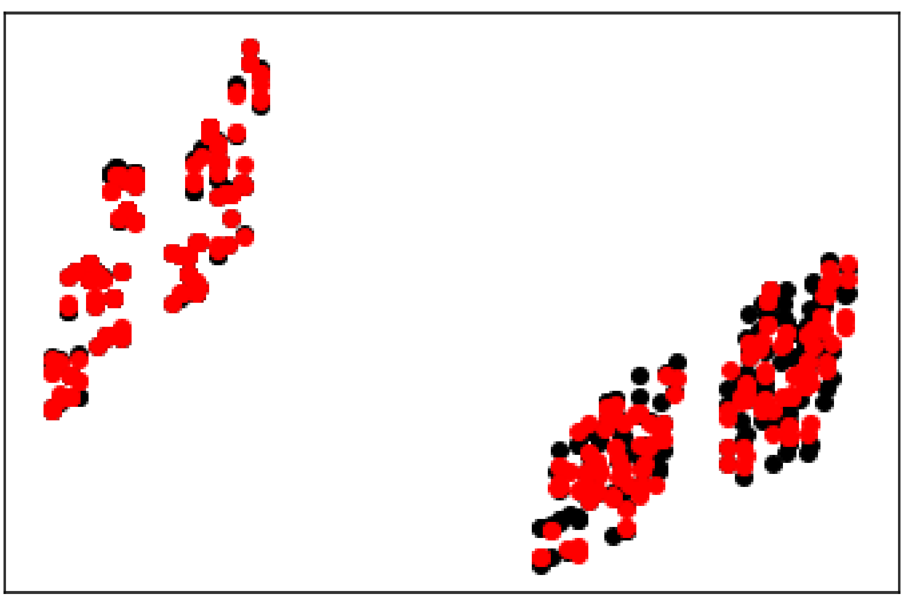
```
svm
Variance score: 0.96
```



```
reg
Variance score: 0.79
```



```
random forest
Variance score: 0.97
```



Group of configurations
—

```
In [9]: classement_general={}
        for j in range(len(listeVideo)):
            classement = {}
            liste_temps=listeVideo[j][predDimension]
            for i in range(len(listeVideo[j][predDimension])):
                classement[listeVideo[j]["configurationID"][i]]=listeVideo[j][predDimension][i]
            classement=sorted(classement.items(), key=lambda t:t[1])
            classement_general[j]=classement
        len(classement_general)

Out[9]: 17
```

```
In [10]: tableau={}
         for c in range(1,len(listeVideo[0])+1):
             conf1={}
             for i in range(len(listeVideo)):
                 classement_config=0
                 for j in range(len(listeVideo[0])):
                     if classement_general[i][j][0]==c:
                         classement_config = classement_general[i].index(classement_general[i][j

                 conf1[i+1] = classement_config
             tableau[c]=conf1
```

Dataframe of ordering configurations
—

```
In [11]: tableau2=pandas.DataFrame(data=tableau)
         tableau_joli=tableau2.transpose()

         ## for each configuration id, ranking for each video
         tableau_joli
         #se lit comme tel : la première configuration est la deuxième moins efficace pour la pr
```

Out[11]:

	1	2	3	4	5	6	7	8	9	10	11	12	\
1	754	889	783	886	831	831	888	552	995	899	831	814	
2	336	517	377	486	441	484	421	241	460	403	410	414	
3	423	437	512	456	482	467	428	628	604	335	344	333	
4	184	139	169	128	176	161	81	369	303	164	114	154	
5	1114	1047	1124	1076	1116	1039	1142	1129	1123	949	1057	1057	
6	238	312	217	269	255	234	254	109	240	310	243	316	
7	729	886	651	876	844	860	838	544	480	730	713	748	
8	802	628	844	678	688	575	713	957	927	660	810	779	
9	835	845	896	816	879	797	921	832	889	598	621	637	

10	1029	1093	1090	1088	1029	1090	1069	975	1087	1121	1103	1098
11	119	159	179	199	199	152	164	439	380	190	182	182
12	449	551	438	511	518	535	481	291	387	519	506	477
13	772	690	823	730	661	804	671	633	790	996	868	899
14	1111	1134	1085	1130	1093	1083	1079	1037	1108	1045	1083	1068
15	486	409	470	393	413	409	551	228	253	720	540	464
16	350	241	261	286	287	253	300	63	92	366	337	233
17	364	248	314	238	263	286	267	54	267	646	363	421
18	840	736	789	811	687	766	653	789	835	827	865	845
19	673	668	590	589	681	665	677	484	497	912	700	792
20	1060	1020	1045	1001	1091	1025	1042	1149	1030	939	1016	1024
21	581	548	597	578	592	489	576	894	612	380	692	572
22	120	142	118	136	188	109	159	322	136	79	105	83
23	788	734	635	781	641	764	473	786	523	732	855	850
24	924	946	899	948	952	981	908	706	735	1003	919	893
25	750	613	774	638	656	594	609	966	756	551	775	739
26	980	1066	1000	1044	965	1091	925	940	973	1102	1097	1089
27	982	980	992	980	1019	932	1017	1080	1023	756	976	944
28	839	633	810	680	702	567	696	937	897	524	819	689
29	246	305	231	276	236	232	234	99	68	302	299	278
30	1139	1040	1105	1053	1119	1043	1118	1115	1143	787	1040	1010
...	...	...	...	...	...	...	...	...	...	...	...	...
1123	143	169	174	147	169	159	158	436	374	179	108	151
1124	112	181	122	191	216	120	240	407	324	45	135	130
1125	100	112	141	107	144	187	58	374	207	174	106	157
1126	396	479	450	495	500	426	590	595	648	226	325	303
1127	770	566	719	617	645	628	451	933	802	638	760	781
1128	123	141	152	120	149	178	44	363	254	227	123	173
1129	27	73	6	76	91	28	128	201	50	8	7	1
1130	61	78	2	81	49	5	117	211	135	28	47	15
1131	51	71	1	71	59	2	98	204	94	9	22	11
1132	572	541	569	569	567	491	588	900	654	323	658	568
1133	319	432	422	433	459	447	334	624	509	344	322	355
1134	258	386	277	383	390	298	374	473	473	156	237	195
1135	973	986	981	978	1020	945	1020	1088	1050	705	963	914
1136	672	817	728	733	845	773	807	807	839	572	598	613
1137	1112	1049	1125	1066	1122	1047	1141	1130	1140	964	1060	1022
1138	546	792	594	710	805	676	834	697	672	300	470	360
1139	1023	1097	1098	1108	1062	1096	1083	996	1061	1107	1082	1123
1140	1030	1076	1050	1082	1046	1115	1003	974	979	1143	1131	1132
1141	458	526	498	497	511	578	420	281	471	792	539	623
1142	422	607	404	512	488	555	359	419	367	539	567	544
1143	342	511	419	480	434	519	380	282	436	580	402	542
1144	1087	1128	1035	1111	1082	1118	1012	1036	925	1101	1102	1116
1145	226	303	235	255	238	236	250	102	229	280	298	245
1146	781	876	736	840	843	892	808	535	547	934	743	867
1147	704	875	781	831	815	839	847	557	722	723	646	780
1148	893	747	801	833	736	835	665	828	614	930	918	903

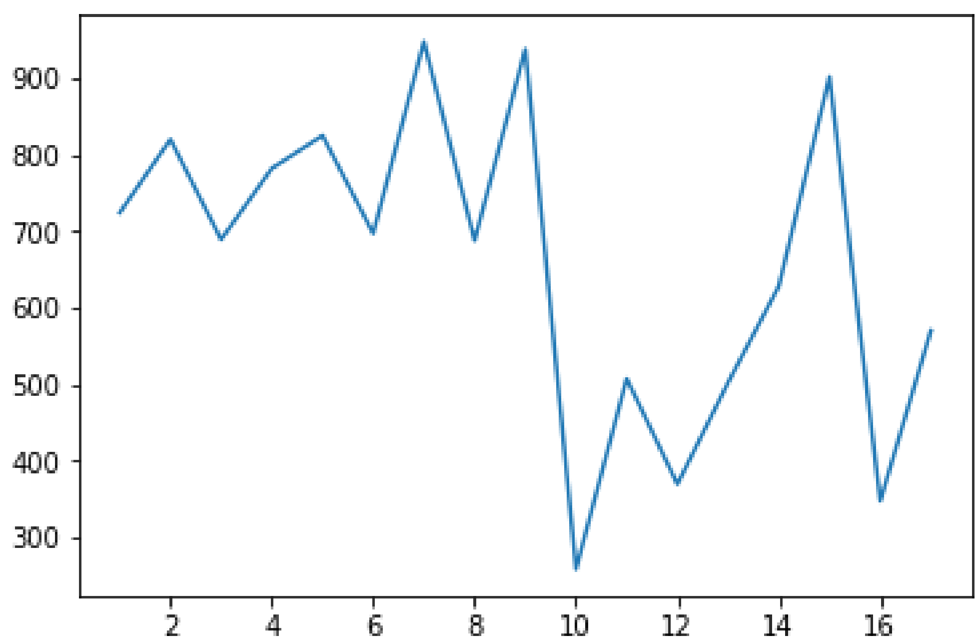
1149	348	240	321	251	273	274	268	79	231	557	326	466
1150	414	362	330	324	302	311	311	232	170	465	387	433
1151	786	697	842	758	659	806	681	742	764	997	876	890
1152	210	213	229	204	105	223	175	28	18	393	220	298

	13	14	15	16	17
1	908	690	870	834	542
2	365	465	463	318	272
3	378	452	488	337	612
4	136	184	190	184	235
5	1059	1106	1129	995	1130
6	198	228	195	238	209
7	767	719	836	771	522
8	728	862	734	640	949
9	752	660	905	579	833
10	1121	1033	1049	1116	1046
11	141	170	251	133	205
12	458	526	520	490	340
13	832	833	671	809	842
14	1093	1109	1118	1013	1041
15	510	374	351	821	396
16	379	271	299	317	175
17	375	273	258	611	289
18	812	909	711	718	810
19	771	570	557	876	515
20	1007	1068	1047	976	1112
21	436	807	643	481	717
22	64	173	176	48	129
23	786	904	645	732	793
24	918	823	913	1083	722
25	641	866	668	626	945
26	1077	1020	983	1087	1053
27	937	1017	1025	866	981
28	719	886	705	606	910
29	308	251	193	286	231
30	1057	1046	1100	943	1074
...	...	...	...	...	...
1123	130	111	263	161	208
1124	110	107	236	39	113
1125	104	191	174	157	199
1126	362	494	536	296	523
1127	625	845	622	627	944
1128	135	155	168	203	193
1129	43	55	139	13	13
1130	21	35	149	35	40
1131	112	49	140	28	30
1132	442	786	636	345	708
1133	352	463	429	346	622

```
1134   286   429   451   126   373
1135   951  1006  1016   803   987
1136   553   626   841   612   824
1137  1051  1080  1126  1001  1120
1138   516   610   834   340   589
1139  1116  1049  1097  1071  1036
1140  1118  1069  1009  1125  1099
1141   507   500   426   742   448
1142   437   515   470   379   410
1143   394   439   419   411   333
1144  1090  1129  1084  1090  1091
1145   275   227   209   311   245
1146   815   677   791   940   645
1147   764   647   807   751   560
1148   869   935   737   918   889
1149   355   268   283   551   282
1150   403   288   331   456   321
1151   875   867   714   928   852
1152   223   181    63   289   163

[1152 rows x 17 columns]
```

In [12]:
```
print ("configuration with the worst std")
tableau_joli.loc[tableau_joli.transpose().describe().transpose()['std'].idxmax()].plot(
print ("std per configuration")
stds_confs = tableau_joli.transpose().describe().transpose()['std'].describe()
```

```
configuration with the worst std
std per configuration
```



In [ ]:

In [13]:
```
groupe={}
for i in range(1,len(listeVideo)):
    groupe[i]=tableau_joli.loc[tableau_joli[i]<10].index
groupe
```

Out[13]:
```
{1: Int64Index([102, 207, 349, 490, 687, 753, 756, 781, 866, 966], dtype='int64'),
 2: Int64Index([102, 428, 529, 537, 574, 605, 608, 651, 921, 1019], dtype='int64'),
 3: Int64Index([57, 262, 349, 436, 908, 958, 1088, 1129, 1130, 1131], dtype='int64'),
 4: Int64Index([88, 340, 386, 428, 529, 537, 608, 651, 876, 996], dtype='int64'),
 5: Int64Index([88, 257, 386, 574, 580, 643, 651, 685, 692, 996], dtype='int64'),
 6: Int64Index([133, 156, 430, 866, 875, 918, 958, 1088, 1130, 1131], dtype='int64'),
 7: Int64Index([88, 257, 386, 449, 529, 574, 580, 605, 651, 876], dtype='int64'),
 8: Int64Index([163, 199, 238, 290, 654, 736, 822, 869, 960, 1099], dtype='int64'),
 9: Int64Index([224, 420, 507, 548, 720, 736, 782, 784, 822, 869], dtype='int64'),
 10: Int64Index([35, 133, 356, 424, 436, 490, 908, 958, 1129, 1131], dtype='int64'),
 11: Int64Index([35, 48, 349, 441, 580, 753, 781, 839, 918, 1129], dtype='int64'),
 12: Int64Index([80, 133, 356, 436, 866, 908, 918, 958, 1088, 1129], dtype='int64'),
 13: Int64Index([35, 207, 430, 485, 490, 744, 843, 973, 1019, 1088], dtype='int64'),
 14: Int64Index([207, 374, 535, 574, 584, 605, 616, 674, 716, 973], dtype='int64'),
 15: Int64Index([56, 199, 340, 386, 428, 449, 537, 574, 903, 1099], dtype='int64'),
 16: Int64Index([35, 232, 423, 436, 490, 754, 839, 866, 918, 1088], dtype='int64')}
```

Top 10 configurations

–

In [14]:
```
groupe_config={}
for i in (0,9):
    for j in range(1,len(listeVideo)):
        l=[]
        for c in range(0,10):
            for k in range(1,len(listeVideo)):
                if groupe[j][i]==groupe[k][c]:
                    if groupe[j][i] not in groupe_config.keys():
                        l.append(k)
                        groupe_config[groupe[j][i]]=l
                    else :
                        groupe_config[groupe[j][i]].append(k)
for i in groupe_config:
    groupe_config[i]=set(groupe_config[i])
groupe_config
```

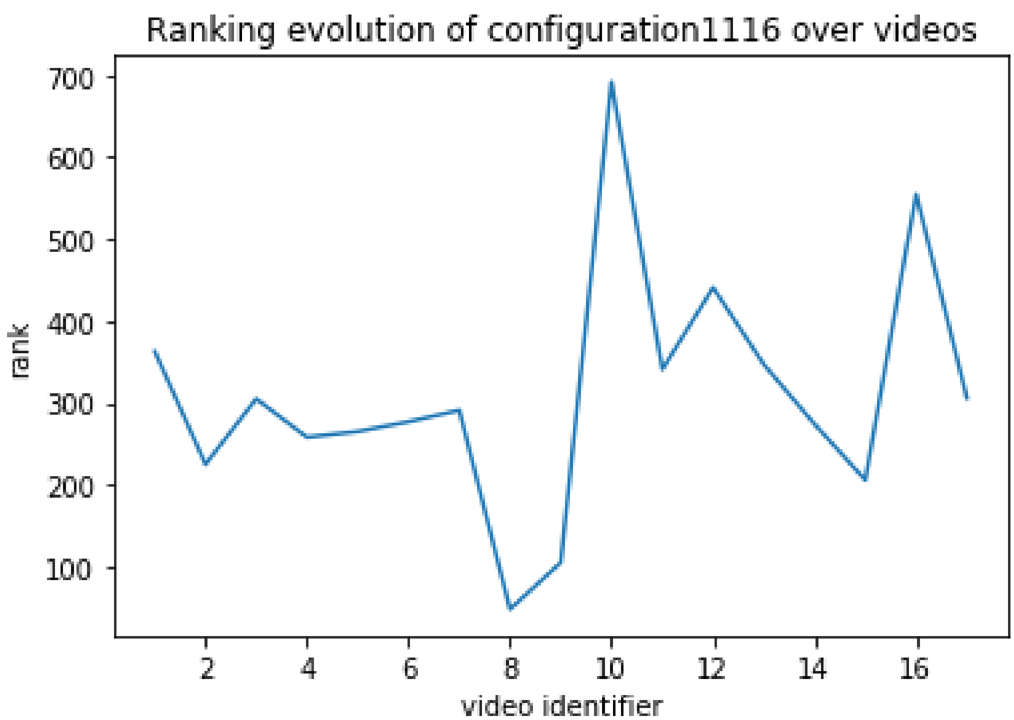
Out[14]:
```
{102: {1, 2},
 57: {3},
 88: {4, 5, 7},
```

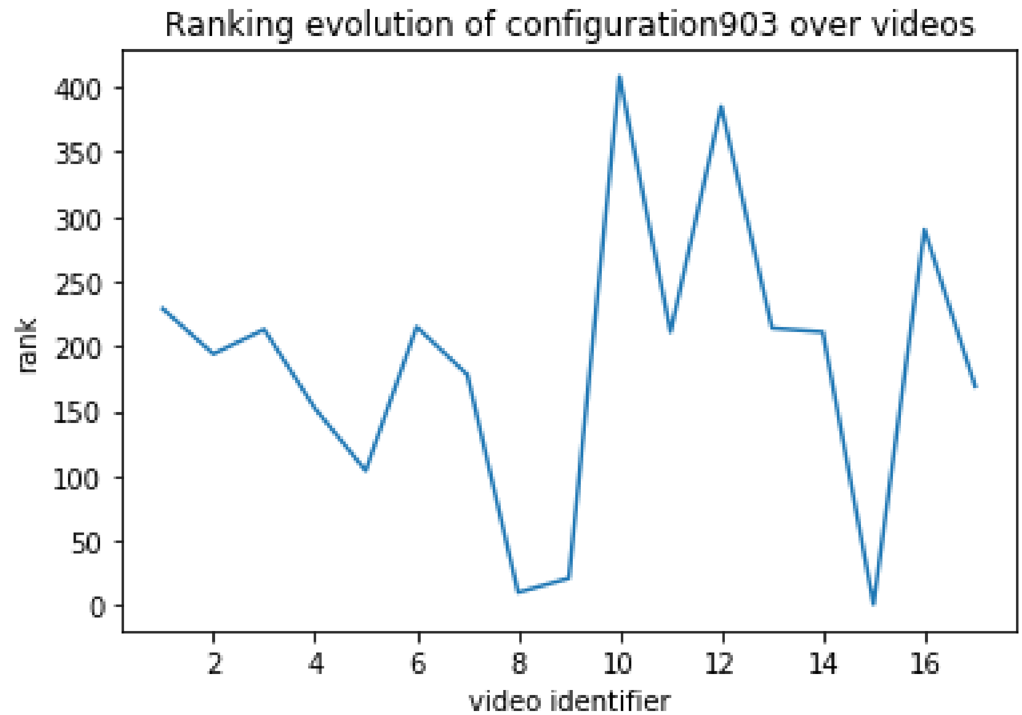
```
 133: {6, 10, 12},
 163: {8},
 224: {9},
 35: {10, 11, 13, 16},
 80: {12},
 207: {1, 13, 14},
 56: {15},
 966: {1},
 1019: {2, 13},
 1131: {3, 6, 10},
 996: {4, 5},
 876: {4, 7},
 1099: {8, 15},
 869: {8, 9},
 1129: {3, 10, 11, 12},
 1088: {3, 6, 12, 13, 16},
 973: {13, 14}}
```

In [15]:
```
import pandas as pd
rank_configs = tableau_joli.transpose().describe(percentiles=[.1, .25, .5, .75, .9]).tr
rank_maxmin_diff = pd.Series(rank_configs['max'] - rank_configs['min']).idxmax() # 1114
print("Worst case rank diff (with at least one ranking in top 100) " + str(rank_maxmin_
# worstcase_rank_diff = tableau_joli.transpose()[worstcase_rank_maxmin_diff].values.arg
# tableau_joli.transpose()[worstcase_rank_diff].describe()
#tableau_joli.transpose()[1114].plot()
#plt.show()

#(rank_configs['mean']).argmax()
#tableau_joli.transpose()[404].describe()
#rank_configs['std'].sort_values()
#(rank_configs['25%'] - rank_configs['75%']).sort_values() #.describe()
#(rank_configs['10%'] - rank_configs['90%']).sort_values()
worst_config_rank = (rank_configs['10%'] - rank_configs['50%']).sort_values().index[0]
print("worst_config_rank dispersion (between 10% and 50%)" + str(worst_config_rank))
#(rank_configs['max'] - rank_configs['min']).sort_values()

def rank_evolution(cid):
    tableau_joli.transpose()[cid].plot()
    plt.xlabel('video identifier')
    plt.ylabel('rank')
    plt.title("Ranking evolution of configuration" + str(cid) + " over videos")
    plt.savefig("rankingevo-c" + str(cid) + ".pdf", format="pdf", bbox_inches='tight')
    plt.show()




# huge fluctuations (but on the overall)
```

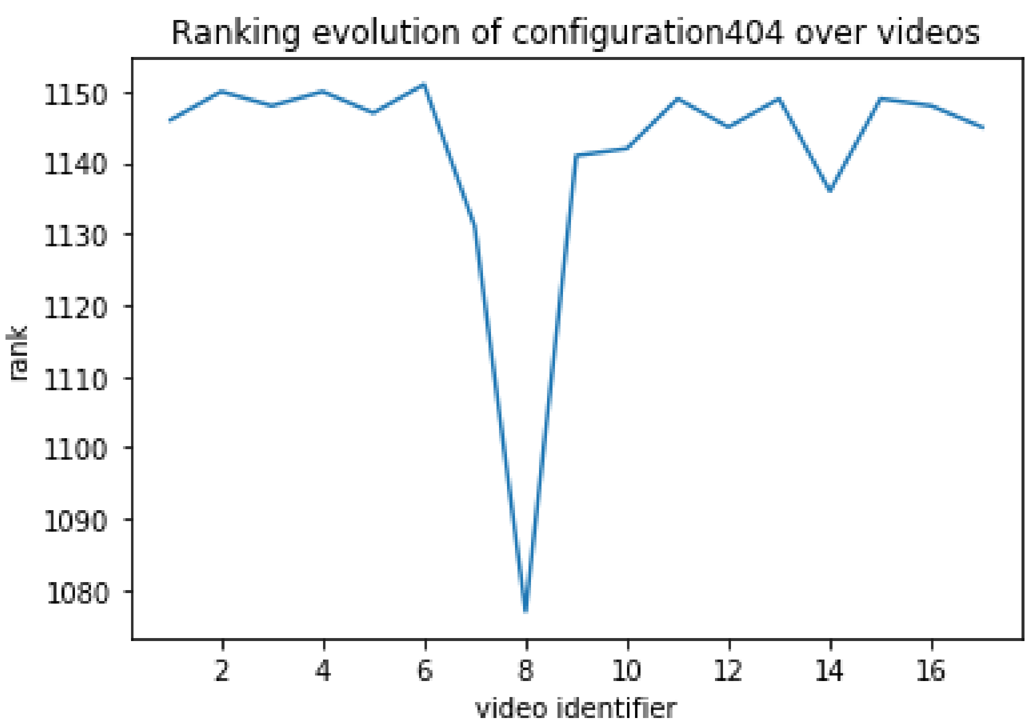
```python
rank_evolution(rank_maxmin_diff)
rank_min_diff = tableau_joli.transpose()[rank_maxmin_diff].min()
rank_max_diff = tableau_joli.transpose()[rank_maxmin_diff].max()
video_rank_max_diff = tableau_joli.transpose()[rank_maxmin_diff].idxmax()
video_rank_min_diff = tableau_joli.transpose()[rank_maxmin_diff].idxmin()
video_rank_max_diff = tableau_joli.transpose()[rank_maxmin_diff].idxmax()

# huge fluctuations
rank_evolution(worst_config_rank)

# small fluctuations (eg always a worst configuration)
rank_evolution(tableau_joli.transpose().describe().transpose()['std'].idxmin())
#tableau_joli.transpose().describe().transpose().describe()

rank_evolution(tableau_joli.transpose().describe().transpose()['mean'].idxmin())
```

```
Worst case rank diff (with at least one ranking in top 100) 1116
worst_config_rank dispersion (between 10% and 50%)903
```

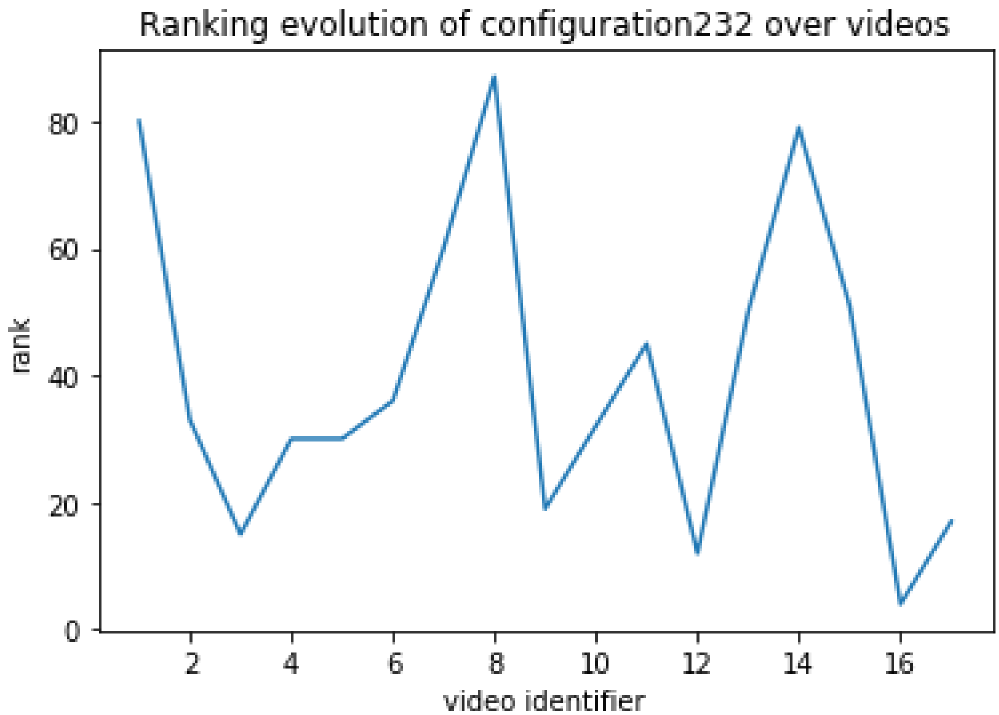




Ranking evolution of configuration903 over videos
rank
400
350
300
250
200
150
100
50
0
2
4
6
8
10
12
14
16
video identifier



Ranking evolution of configuration404 over videos
rank
1150
1140
1130
1120
1110
1100
1090
1080
2
4
6
8
10
12
14
16
video identifier



1 Are there some configurations more sensitive to input videos?

The standard deviation among ranking configurations is 96.08 on average (max: 208.46). There are cases in which configurations have a stable ranking: It lets suggest that ranking changes per configuration are not significant (in general). However there are less favorable cases. Configuration 1116 is ranked 48th for video 8 and 692th for video 10 (out of 1052). This "swing" is the most important one.

Figure below shows another configuration example with noticeable changes in the rankings: In practical terms, the reuse of performance prediction model for some configurations and some videos can lead to the choice of suboptimal configurations.

```
In [16]: rank_maxmin_diff

Out[16]: 1116

In [17]: tableau_joli.transpose().describe().transpose().query('min == 0')['max'].argmax()

/usr/local/lib/python3.7/site-packages/ipykernel_launcher.py:1: FutureWarning: 'argmax' is depre
will be corrected to return the positional maximum in the future.
Use 'series.values.argmax' to get the position of the maximum now.
  """Entry point for launching an IPython kernel.


Out[17]: 1099
```

In [18]:
```
import scipy
tableau_joli.transpose()[224].argmax(), tableau_joli.transpose()[224].argmin()
tableau_joli.transpose()[224][6], tableau_joli.transpose()[224][13]
np.corrcoef(listeVideo[5][predDimension], listeVideo[13][predDimension])[0, 1], scipy.s
```

```
/usr/local/lib/python3.7/site-packages/ipykernel_launcher.py:2: FutureWarning: 'argmax' is depre
will be corrected to return the positional maximum in the future.
Use 'series.values.argmax' to get the position of the maximum now.

/usr/local/lib/python3.7/site-packages/ipykernel_launcher.py:2: FutureWarning: 'argmin' is depre
will be corrected to return the positional minimum in the future.
Use 'series.values.argmin' to get the position of the minimum now.
```

Out[18]:
```
(0.9393662916640168,
 SpearmanrResult(correlation=0.9448141971055567, pvalue=0.0))
```

In [19]:
```
tableau_joli.transpose().describe().transpose().query('min < 10')['max'].argmax()
```

```
/usr/local/lib/python3.7/site-packages/ipykernel_launcher.py:1: FutureWarning: 'argmax' is depre
will be corrected to return the positional maximum in the future.
Use 'series.values.argmax' to get the position of the maximum now.
  """Entry point for launching an IPython kernel.
```

Out[19]:
```
420
```

In [20]:
```
tableau_joli.transpose()[44].argmax(), tableau_joli.transpose()[44].argmin()
tableau_joli.transpose()[44][6], tableau_joli.transpose()[44][1]
```

```
/usr/local/lib/python3.7/site-packages/ipykernel_launcher.py:1: FutureWarning: 'argmax' is depre
will be corrected to return the positional maximum in the future.
Use 'series.values.argmax' to get the position of the maximum now.
  """Entry point for launching an IPython kernel.
/usr/local/lib/python3.7/site-packages/ipykernel_launcher.py:1: FutureWarning: 'argmin' is depre
will be corrected to return the positional minimum in the future.
Use 'series.values.argmin' to get the position of the minimum now.
  """Entry point for launching an IPython kernel.
```

Out[20]:
```
(257, 244)
```

In [21]:
```
tableau_joli.transpose().describe(percentiles=[.05]).transpose()['5%'].argmin()
# tableau_joli.transpose()[1088].describe() (good for top 25%)
# tableau_joli.transpose()[163].describe() # configuration 163 (top 10%)
# tableau_joli.transpose()[580].describe() # configuration 580 (top 5%)
# tableau_joli.transpose().describe().transpose()['mean'].argmin()
# tableau_joli.transpose()[839].describe()
# tableau_joli.transpose().describe().transpose()['std'].argmax()
#tableau_joli.transpose()[419].describe()
```

```
/usr/local/lib/python3.7/site-packages/ipykernel_launcher.py:1: FutureWarning: 'argmin' is depre
will be corrected to return the positional minimum in the future.
Use 'series.values.argmin' to get the position of the minimum now.
  """Entry point for launching an IPython kernel.
```

```
Out[21]: 490
```

```
In [22]: tableau_joli.transpose().describe().transpose().query('min == 0')['mean'].argmax()
         #tableau_joli.transpose()[44].describe()
         # tableau_joli.transpose()[163].describe()
         tableau_joli.transpose()[224].describe()
```

```
/usr/local/lib/python3.7/site-packages/ipykernel_launcher.py:1: FutureWarning: 'argmax' is depre
will be corrected to return the positional maximum in the future.
Use 'series.values.argmax' to get the position of the maximum now.
  """Entry point for launching an IPython kernel.
```

```
Out[22]: count     17.000000
         mean     192.411765
         std      122.480641
         min        6.000000
         25%      172.000000
         50%      197.000000
         75%      221.000000
         max      518.000000
         Name: 224, dtype: float64
```

```
In [23]: # video 2 and video 5
         import pandas as pd
         (tableau_joli[3] - tableau_joli[6]).describe() # 3 because we are staring from 1 (so vi
         # tableau_joli[3].index[tableau_joli[3] == 0]
         # tableau_joli[3].argmin()
         #tableau_joli[3].index[tableau_joli[3] < 10]
         tableau_joli[3][256] - tableau_joli[6][256]
         # tableau_joli[3].nlargest(10)
         #(abs(tableau_joli[3] - tableau_joli[15])).describe() # good Spearman correlation
         # tableau_joli[15][tableau_joli[3].argmin()], tableau_joli[3][tableau_joli[15].argmin()
         # tableau_joli[15][tableau_joli[3].index[tableau_joli[3] == 3]]

         ### eg top 10 configurations of video 2 vs video 14
         ### we start at 1, grrrr TODO
         def diff_rank_top(v1ID, v2ID):
             rv1ID = v1ID + 1 # because we start at 1, grrrr TODO
             rv2ID = v2ID + 1
             rankBy = tableau_joli.sort_values(by=rv1ID, axis=0)
             m = pd.concat([rankBy[rv1ID][:10], rankBy[rv2ID][:10]], axis=1)
             m.columns = ['video ' + str(v1ID), 'video ' + str(v2ID)]
```

```
    return m
#diff_rank_top(2, 0)
#diff_rank_top(2, 14)
diff_rank_top(14, 9)
```

Out[23]:

	video 14	video 9
1099	0	384
903	1	408
199	2	386
574	3	107
537	4	98
340	5	137
56	6	121
428	7	120
449	8	129
386	9	152

In [24]:
```
tableau_joli.transpose().describe().transpose().sort_values(by="std")
```

Out[24]:

	count	mean	std	min	25%	50%	75%	max
404	17.0	1141.411765	17.435807	1077.0	1142.0	1147.0	1149.0	1151.0
526	17.0	1138.117647	18.694526	1079.0	1137.0	1145.0	1151.0	1151.0
515	17.0	1136.352941	21.207137	1076.0	1138.0	1144.0	1148.0	1150.0
444	17.0	1122.705882	22.552064	1048.0	1119.0	1125.0	1134.0	1145.0
297	17.0	1136.235294	24.061196	1072.0	1141.0	1145.0	1149.0	1150.0
476	17.0	1089.058824	24.961146	1035.0	1067.0	1099.0	1103.0	1125.0
232	17.0	40.000000	24.974987	4.0	19.0	33.0	51.0	87.0
1048	17.0	1085.117647	25.859917	1010.0	1068.0	1095.0	1102.0	1110.0
281	17.0	1084.352941	25.995050	1039.0	1064.0	1089.0	1101.0	1125.0
219	17.0	51.117647	26.009331	1.0	33.0	49.0	68.0	98.0
145	17.0	1110.529412	26.287159	1027.0	1100.0	1119.0	1127.0	1136.0
344	17.0	1112.764706	26.947935	1023.0	1106.0	1121.0	1129.0	1142.0
1010	17.0	1134.117647	27.253170	1038.0	1133.0	1143.0	1149.0	1151.0
611	17.0	48.470588	27.388678	10.0	25.0	41.0	65.0	108.0
564	17.0	1121.000000	27.669930	1044.0	1118.0	1130.0	1139.0	1147.0
587	17.0	57.058824	27.734163	10.0	42.0	50.0	65.0	125.0
646	17.0	1135.411765	27.861395	1032.0	1138.0	1142.0	1148.0	1151.0
62	17.0	1116.411765	27.959924	1047.0	1110.0	1117.0	1140.0	1147.0
955	17.0	1115.588235	28.651045	1015.0	1114.0	1117.0	1130.0	1141.0
939	17.0	1077.588235	28.727293	1005.0	1071.0	1086.0	1095.0	1108.0
1041	17.0	1113.235294	28.784825	1022.0	1104.0	1119.0	1131.0	1143.0
115	17.0	1129.882353	28.841988	1031.0	1131.0	1139.0	1146.0	1148.0
411	17.0	1115.235294	28.910486	1045.0	1107.0	1124.0	1138.0	1146.0
790	17.0	1072.882353	29.274311	1002.0	1057.0	1078.0	1096.0	1114.0
305	17.0	1131.647059	29.635159	1033.0	1135.0	1142.0	1146.0	1150.0
976	17.0	1117.176471	30.170837	1018.0	1119.0	1126.0	1134.0	1141.0
629	17.0	1115.941176	30.468161	1020.0	1117.0	1123.0	1132.0	1143.0
450	17.0	1068.411765	30.577890	1000.0	1053.0	1079.0	1092.0	1114.0

```
808    17.0   49.941176   31.423460   11.0   26.0   45.0   62.0   122.0
349    17.0   41.705882   31.570882    0.0   14.0   45.0   58.0    93.0
...     ...         ...         ...    ...    ...    ...    ...     ...
771    17.0  754.588235  172.403980  389.0  674.0  800.0  855.0   969.0
65     17.0  434.647059  174.532211   89.0  344.0  389.0  487.0   791.0
1079   17.0  748.117647  174.653186  407.0  623.0  815.0  863.0  1009.0
380    17.0  411.882353  175.005315   66.0  324.0  392.0  462.0   782.0
291    17.0  493.058824  175.612311  232.0  376.0  474.0  572.0   853.0
1111   17.0  407.411765  175.677282   96.0  327.0  393.0  491.0   801.0
106    17.0  612.117647  175.682911  279.0  483.0  619.0  731.0   849.0
951    17.0  749.352941  175.687557  412.0  683.0  798.0  862.0  1002.0
470    17.0  492.411765  175.711076  260.0  373.0  475.0  521.0   880.0
367    17.0  402.000000  177.027540   85.0  321.0  367.0  486.0   778.0
925    17.0  596.588235  177.383785  288.0  478.0  618.0  712.0   840.0
633    17.0  772.588235  177.464454  379.0  680.0  816.0  869.0  1023.0
239    17.0  413.058824  177.871608  113.0  323.0  355.0  506.0   774.0
122    17.0  643.176471  178.833804  314.0  575.0  687.0  751.0   900.0
1004   17.0  511.529412  181.073230  287.0  377.0  472.0  571.0   909.0
159    17.0  661.176471  182.560002  327.0  518.0  710.0  807.0   913.0
172    17.0  428.647059  184.198582  123.0  311.0  358.0  506.0   814.0
302    17.0  660.705882  184.810296  282.0  531.0  693.0  812.0   943.0
283    17.0  655.294118  185.068151  292.0  555.0  701.0  802.0   909.0
1017   17.0  433.000000  185.973788  114.0  299.0  395.0  518.0   850.0
322    17.0  669.823529  188.063897  305.0  596.0  708.0  803.0   957.0
119    17.0  677.000000  188.240870  298.0  571.0  696.0  815.0   947.0
985    17.0  756.235294  190.291398  338.0  658.0  814.0  870.0  1008.0
383    17.0  650.117647  192.630113  289.0  498.0  672.0  799.0   950.0
85     17.0  653.294118  196.131259  268.0  520.0  698.0  808.0   926.0
1119   17.0  747.823529  197.371235  370.0  664.0  805.0  863.0  1005.0
419    17.0  657.470588  197.789066  315.0  573.0  689.0  809.0   944.0
1114   17.0  415.823529  200.920767  119.0  312.0  400.0  477.0   974.0
459    17.0  666.823529  202.716315  269.0  579.0  692.0  821.0   954.0
486    17.0  658.470588  208.461962  258.0  507.0  689.0  820.0   948.0

[1152 rows x 8 columns]
```

In [25]:
```python
ranking_general_size={}
for j in range(len(listeVideo)):
    ranking_size = {}
    # liste_size=listeVideo[j]["size"]
    for i in range(len(listeVideo[j]["size"])):
        ranking_size[listeVideo[j]["configurationID"][i]]=listeVideo[j]["size"][i]
    ranking_size=sorted(ranking_size.items(), key=lambda t:t[1])
    ranking_general_size[j]=ranking_size
len(ranking_general_size)
```

Out[25]: 17

In [26]:
```python
tableau_size={}
```

```
for c in range(1,len(listeVideo[0])+1):
    conf1_size={}
    for i in range(len(listeVideo)):
        classement_config_size=0
        for j in range(len(listeVideo[0])):
            if ranking_general_size[i][j][0]==c:
                classement_config_size = ranking_general_size[i].index(ranking_general_

        conf1_size[i+1] = classement_config_size
    tableau_size[c]=conf1_size
```

In [27]:
```
tableau3=pandas.DataFrame(data=tableau_size)
tableau_joli_size=tableau3.transpose()
tableau_joli_size
```

Out[27]:
```
         1    2     3    4    5     6     7     8     9    10   11    12  \
1      127  665    29  462  374   598    98    39    75   655  693   627
2      133  671    53  517  366   607   112    61    72   659  717   649
3     1117  505  1084  638  607   499   970   964  1135   526  502   508
4     1106  472  1112  671  657   530   949  1100  1082   560  494   542
5     1123  490   977  602  700   448  1108   985  1117   406  460   421
6      227  782   161  842  725   805   147   361    84   780  818   859
7      148  716   127  494  599   646   162    94   184   687  690   637
8     1127  515  1070  663  707   470  1124  1043  1115   458  473   464
9     1103  494  1031  597  612   491   980   929  1139   515  479   488
10     350  883   374  874  943   825   401   205   401   923  875   820
11    1146  533  1146  766  760   552  1145  1140  1128   552  549   552
12     159  714   185  559  600   666   190   117   150   723  703   661
13     371  894   534  896  945   868   429   260   421   970  888   862
14     162  730    37  733  673   691   150   148    95   616  744   679
15     165  796   439  824  796  1036   181   374   302   834  777   892
16     214  692   245  647  629   752   200   333   187   781  711   752
17     425  912   698  953  964   984   422   478   419  1082  931  1046
18     173  751    77  780  678   731   160   178    94   657  757   701
19     129  829   372  798  802   959   206   180   315   769  801   811
20    1093  487   962  581  665   445  1072   988   976   397  448   430
21     731   65   326   68   83    32   671   665   591    44   32    38
22     688  184   725  223  205   235   686   835   787   253  226   331
23     122  749    65  757  617   721    79   161    41   652  755   719
24     132  830   280  776  803   913   173   154   319   750  809   788
25    1120  520  1051  627  675   478  1093  1033  1045   445  466   472
26     314  880   322  867  928   822   242   177   276   925  869   824
27     725   29   216   53   80    26   637   626   576    23   23    17
28     616  160   530  154  181   175   747   706   838   178  142   178
29     196  778   157  836  685   802    90   344    64   775  820   850
30     628  145   414  124  187   154   759   658   832   124  109   160
...    ...  ...   ...  ...  ...   ...   ...   ...   ...   ...  ...   ...
1123  1147  534  1147  767  761   553  1146  1141  1129   553  550   553
```

1124	800	242	819	272	284	272	986	1016	842	248	275	314
1125	961	463	1099	620	641	523	919	1081	1003	547	511	535
1126	751	268	751	256	259	256	1009	811	970	214	283	235
1127	918	450	979	505	528	459	858	942	918	408	444	468
1128	962	464	1100	621	642	524	920	1082	1004	548	512	536
1129	832	98	571	148	127	121	736	880	617	127	121	109
1130	833	99	572	149	128	122	737	881	618	128	122	110
1131	834	100	573	150	129	123	738	882	619	129	123	111
1132	730	64	325	67	82	31	670	664	590	43	31	37
1133	1082	527	1064	610	588	494	887	962	1055	524	485	506
1134	795	129	455	111	105	99	696	669	663	90	117	78
1135	724	28	215	52	79	25	636	625	575	22	22	16
1136	1070	503	1018	568	591	485	872	926	1067	497	500	482
1137	1122	489	976	601	699	447	1107	984	1116	405	459	420
1138	786	120	382	93	102	102	655	630	690	75	111	54
1139	451	935	467	903	1005	832	557	327	447	920	889	831
1140	293	889	420	878	949	852	318	206	343	858	868	835
1141	313	964	605	941	952	866	320	251	363	943	960	952
1142	211	808	128	782	680	769	93	166	67	800	819	793
1143	465	967	409	950	907	771	416	304	293	963	996	939
1144	238	761	138	736	813	828	218	224	202	748	747	682
1145	241	777	155	810	689	813	89	351	58	841	825	849
1146	306	954	516	916	950	830	283	183	325	909	946	930
1147	445	959	299	915	906	745	394	255	282	949	974	915
1148	247	793	233	784	815	878	237	273	203	797	764	710
1149	539	978	797	1006	1039	1003	625	543	476	1081	982	1061
1150	278	833	438	847	835	1033	258	450	257	871	838	887
1151	477	947	621	947	1015	897	609	407	489	972	905	865
1152	487	929	514	982	911	809	412	517	254	1026	959	1021

	13	14	15	16	17
1	297	497	70	287	12
2	324	542	97	311	37
3	1074	667	966	782	951
4	1010	676	1091	860	1120
5	817	706	1045	638	900
6	559	389	252	817	257
7	307	495	135	300	77
8	942	770	1082	708	981
9	1022	626	926	735	869
10	634	844	491	780	284
11	1086	821	1146	906	1145
12	334	541	188	368	120
13	666	911	579	864	520
14	322	344	111	485	43
15	467	266	223	890	668
16	353	558	308	500	448
17	871	1042	720	989	972

```
18     408   393   150   556    62
19     463   223   151   726   245
20     721   688  1034   608   897
21      70   364   467    44   408
22     403    64   528   325   720
23     350   395   127   561    64
24     444   186   107   697   180
25     875   772  1057   719   983
26     593   845   449   770   285
27      38   254   396    23   332
28     221    46   366   166   465
29     519   384   221   824   252
30     145    16   335   121   395
...    ...   ...   ...   ...   ...
1123  1087   822  1147   907  1146
1124   478   122   757   347   845
1125   887   697  1066   798  1111
1126   501    94   611   254   585
1127   690   636   907   644   923
1128   888   698  1067   799  1112
1129   249   434   675   157   687
1130   250   435   676   158   688
1131   251   436   677   159   689
1132    69   363   466    43   407
1133   979   662   949   790   955
1134   228   405   488    93   436
1135    37   253   395    22   331
1136   904   614   906   739   872
1137   816   705  1044   637   899
1138   217   276   445    72   360
1139   662   983   655   810   329
1140   596   850   471   809   375
1141   797   948   534   951   538
1142   513   324   147   647    68
1143   963  1117   623   925   276
1144   286   333   210   512   119
1145   508   352   242   767   251
1146   771   897   476   931   306
1147   894  1031   550   865   164
1148   390   373   288   599   205
1149   992  1129   903  1013  1006
1150   540   356   381   873   630
1151   790  1085   736   878   562
1152   947  1131   769   975   771

[1152 rows x 17 columns]
```

```
In [28]: def rank_size_evolution(cid):
```

```
        tableau_joli_size.transpose()[cid].plot()
        plt.xlabel('video identifier')
        plt.ylabel('rank')
        plt.title("Ranking evolution of configuration" + str(cid) + " over videos (size)")
        # plt.savefig("rankingevo-c" + str(cid) + ".pdf", format="pdf", bbox_inches='tight'

        plt.show()

    rank_configs_size = tableau_joli_size.transpose().describe(percentiles=[.1, .25, .5, .7
    (rank_configs_size['max'] - rank_configs_size['min']).argmax() # 1114
    # tableau_joli_size.transpose()[1114].argmax(), tableau_joli_size.transpose()[1114].des


    #(rank_configs['mean']).argmax()
    #tableau_joli.transpose()[404].describe()
    # rank_configs_size['std'].sort_values()
    #(rank_configs['25%'] - rank_configs['75%']).sort_values() #.describe()
    # (rank_configs_size['10%'] - rank_configs_size['90%']).sort_values()
    # (rank_configs_size['10%'] - rank_configs_size['50%']).sort_values()
    (rank_configs_size['10%'] - rank_configs_size['25%']).sort_values()
    #(rank_configs['max'] - rank_configs['min']).sort_values()

    #rank_size_evolution(655)
    #rank_size_evolution(1110)
    #rank_size_evolution(877)
    #rank_size_evolution(7)
    # rank_size_evolution(161)
    # rank_size_evolution(1109)
    # rank_size_evolution(569)
    # rank_size_evolution(1036) # nice one based on (rank_configs_size['10%'] - rank_config
    # rank_size_evolution(tableau_joli_size.transpose().describe().transpose()['std'].argmi
    #tableau_joli_size.transpose().describe().transpose().describe()
```

```
/usr/local/lib/python3.7/site-packages/ipykernel_launcher.py:10: FutureWarning: 'argmax' is depr
will be corrected to return the positional maximum in the future.
Use 'series.values.argmax' to get the position of the maximum now.
  # Remove the CWD from sys.path while we load stuff.
```

```
Out[28]: 981    -37.2
         947    -37.2
         849    -37.2
         519    -30.4
         757    -29.4
         783    -29.4
         761    -29.4
         899    -26.4
         954    -26.0
         513    -23.8
```

654	-23.6
238	-23.6
395	-21.0
492	-21.0
426	-21.0
543	-20.6
724	-18.2
940	-15.2
840	-13.8
887	-13.8
906	-13.8
935	-13.8
1047	-13.4
736	-12.8
822	-12.8
152	-11.4
874	-11.4
31	-11.4
73	-11.4
658	-8.4
828	-8.4
810	-8.4
842	-8.4
621	-8.4
817	-8.4
873	-8.4
751	-8.4
750	-6.6
360	-6.0
632	-5.2
657	-5.0
626	-4.2
594	-4.2
427	-4.2
706	-4.2
455	-4.2
385	-4.2
241	-4.0
738	-4.0
588	-3.0
804	-3.0
752	-3.0
836	-3.0
860	-1.4
618	-1.2
807	-1.2
653	-1.2
245	-1.0

```
740     -1.0
dtype: float64
```

```
In [29]: import seaborn as sns
         from IPython.display import display, HTML

         nvideos = len(listeVideo)
         rankdiff = [[0 for x in range(nvideos)] for y in range(nvideos)]
         pred_diff = [[0 for x in range(nvideos)] for y in range(nvideos)]
         for vid in range(nvideos):
             rvid = pd.DataFrame(listeVideo[vid][predDimension]).rank()
             amin = rvid[predDimension].values.argmin()
             for i in range(nvideos):
                 if (i != vid):
                     rvidi = pd.DataFrame(listeVideo[i][predDimension]).rank()
                     rankdiff[i][vid] = rvidi.loc[amin][predDimension]

                     argbesti = listeVideo[i][predDimension].values.argmin()
                     besti = listeVideo[i].loc[argbesti][predDimension]
                     bestvid = listeVideo[i].loc[amin][predDimension]
                     pred_diff[i][vid] = (1 - (besti/bestvid)) * 100
                     # abs(bestvid - besti)




         display(HTML(pd.DataFrame(rankdiff).style.set_caption("Best ranking difference").backgr
         display(HTML(pd.DataFrame(pred_diff).style.set_caption("Impact of ranking changes (perc


         #pd.DataFrame(pred_diff).style.set_caption("Impact of ranking changes (percentage incre

         #pd.DataFrame(rankdiff).plot.box()
         #plt.show()




         #pd.DataFrame(listeVideo[0]).sort_values(by=predDimension) #, listeVideo[1]
         #pd.DataFrame(listeVideo[2]).sort_values(by=predDimension)[:100]#.loc[756]
```

```
/usr/local/lib/python3.7/site-packages/matplotlib/__init__.py:886: MatplotlibDeprecationWarning:
examples.directory is deprecated; in the future, examples will be found relative to the 'datapat
  "found relative to the 'datapath' directory.".format(key))
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

In [30]:
```python
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error

def mean_relative_error(y_true, y_pred):
    return np.mean(np.abs((y_true - y_pred) / y_true)) * 100

# reusing prediction model of v1ID for v2ID
def reusePredictionModel(v1ID, v2ID):
    p1 = pd.DataFrame(listeVideo[v1ID]).copy()
    r1 = pd.DataFrame(p1.sort_values(by=predDimension)[predDimension]) # rank(method="m
    #print(r1)
    p2 = pd.DataFrame(listeVideo[v2ID]).copy().sort_values(by=predDimension)
    #print(p2[predDimension][:5])
    #print(p2)
    # transfer rank
    predictedValues = pd.DataFrame(columns=['predicted_' + predDimension, predDimension

    #print(p2)
    ind = 0
    for i, r in r1.iterrows():
        nvalue = p2.iloc[ind][predDimension]
        #print (str(i) + " " + str(nvalue))
        predictedValues.loc[i] = [nvalue, p2.loc[i][predDimension]]
        ind = ind + 1
    # p2['predicted_' + predDimension] = predictedValues
    p2 = predictedValues.sort_values(by=predDimension)
    pdiff = pd.DataFrame(p2)
    pdiff['diff'] = pdiff[predDimension] - pdiff['predicted_' + predDimension]
    return p2, mean_relative_error(pdiff[predDimension], pdiff['predicted_' + predDimen

    #for r in r1:
    #    print(r)

maeij = [[1.0 for x in range(len(listeVideo))] for y in range(len(listeVideo))]
for i in range(len(listeVideo)):
    for j in range(len(listeVideo)):
        if (i !=j):
            p, m = reusePredictionModel(i, j)
            maeij[i][j] = m


pd.DataFrame(maeij)
```

Out[30]:
```
            0           1           2           3           4           5    \
```

0	1.000000	6.126732	1.841275	3.905744	5.524166	4.943798
1	4.483708	1.000000	4.219788	1.923972	2.505864	2.322357
2	1.796691	5.331098	1.000000	3.056194	4.641939	4.283451
3	3.568837	2.294500	2.925159	1.000000	2.828025	2.793125
4	4.108967	3.194081	3.485767	2.346239	1.000000	4.668517
5	3.924768	2.753159	3.683042	2.706152	4.278965	1.000000
6	5.006002	5.653744	5.356116	5.159444	3.371445	6.790426
7	10.653043	13.382691	10.106153	9.785302	10.827589	14.466100
8	8.539073	10.255915	7.716589	7.718359	7.925957	11.152674
9	9.423202	12.553161	11.243012	11.996428	13.156228	9.442413
10	2.933559	7.296616	3.829392	5.031575	7.854122	5.221385
11	4.945085	8.284313	5.873598	6.861911	9.020065	5.699905
12	3.651420	7.205590	4.889545	5.602504	7.788408	5.128512
13	4.039083	7.288348	3.654305	3.641316	7.226900	5.416732
14	4.965761	4.598827	4.207277	3.088158	1.932089	6.295659
15	7.622642	11.044666	9.111203	10.167655	11.080539	8.229240
16	5.498616	9.040990	4.822911	5.716071	7.987918	7.632288

	6	7	8	9	10	11 \
0	5.211908	12.317056	9.560401	9.443113	3.773827	5.780648
1	4.479220	13.677992	9.928376	10.442587	6.571299	7.488671
2	4.937397	11.635026	8.154334	9.741484	4.322787	5.777085
3	4.458476	12.425271	9.023847	10.330761	5.665706	6.806806
4	2.925271	11.846326	8.134891	11.902739	7.163267	8.704965
5	5.799688	14.067871	10.636360	8.354946	5.328174	5.246291
6	1.000000	14.050081	9.376375	12.238882	8.572410	9.796136
7	12.056085	1.000000	7.363074	18.848262	13.545648	15.499672
8	8.479351	8.046298	1.000000	16.379098	11.270836	12.721887
9	11.732279	23.943899	19.323991	1.000000	9.239867	5.470324
10	7.925424	14.652715	11.714210	7.575065	1.000000	3.496713
11	9.270123	17.163087	13.830159	5.377638	3.818709	1.000000
12	7.126083	16.574930	12.976645	6.664433	3.594623	4.108402
13	8.365742	10.788159	9.176791	10.877272	4.209945	6.505939
14	3.216406	10.630037	7.171936	12.952095	8.182995	9.704875
15	10.035875	21.264399	17.150783	3.283083	7.538311	4.734531
16	9.069117	7.511762	8.239603	12.243718	6.841939	8.187256

	12	13	14	15	16
0	4.512758	5.761959	7.511708	7.229046	5.995221
1	7.001838	6.537692	3.602453	8.860909	7.168086
2	5.276543	4.551532	6.154321	7.494333	4.941171
3	5.934729	3.921770	3.834680	8.339867	6.045570
4	7.993133	5.784168	2.078522	9.687131	6.715337
5	5.302345	6.728561	6.209999	6.736573	6.322946
6	8.266891	10.673074	4.257198	10.174760	9.584435
7	16.345543	11.181262	10.255747	16.306817	6.839038
8	12.989628	10.270265	7.484465	13.879454	7.658894
9	7.309179	21.089784	16.628374	3.190482	14.527158

```
10   3.419370   5.776896   9.875466   5.832602   6.288346
11   4.355259  10.442988  11.408810   4.392490   8.349772
12   1.000000   9.870648  10.153777   5.105057   8.540481
13   6.884702   1.000000   8.322727   8.917655   4.325860
14   9.153943   5.579101   1.000000  10.714070   6.472038
15   5.847231  17.877772  14.170160   1.000000  12.198813
16   9.264671   4.990403   8.922291   9.865284   1.000000
```

In [31]:
```python
import pandas as pd
from sklearn import preprocessing

videoID = 8
df = pd.DataFrame(listeVideo[videoID][predDimension])
df

normalizer = preprocessing.Normalizer().fit(df)  # fit does nothing

min_max_scaler = preprocessing.MinMaxScaler()
np_scaled = min_max_scaler.fit_transform(df)
df_normalized = pd.DataFrame(np_scaled, columns=[predDimension])
df.sort_values(by=predDimension)[:5], df_normalized.sort_values(by=predDimension)[:5]

X = []
for i in range(len(listeVideo)):
    X.append(listeVideo[i][predDimension])

norms = pd.DataFrame(preprocessing.normalize(X, norm='max')).transpose()
for i in range(len(listeVideo)):
    listeVideo[i][predDimension] = norms[i]

listeVideo[5]
#np.corrcoef(norms[0], norms[1])[0, 1], np.corrcoef(listeVideo[0][predDimension], liste
```

Out[31]:
```
    configurationID  H264  no_8x8dct  no_asm  no_cabac  no_deblock  \
0                 1  True       True   False     False        True
1                10  True       True   False      True       False
2               100  True       True   False     False        True
3              1000  True       True   False      True       False
4              1001  True      False   False     False        True
5              1002  True      False   False      True       False
6              1003  True      False   False     False        True
7              1004  True      False   False      True       False
8              1005  True       True   False     False        True
9              1006  True       True   False     False        True
10             1007  True       True   False     False        True
11             1008  True      False   False     False        True
12             1009  True      False   False      True        True
13              101  True      False   False     False       False
```

14	1010	True	False	False	True	False
15	1011	True	True	False	True	True
16	1012	True	False	False	True	True
17	1013	True	False	False	True	True
18	1014	True	False	False	True	True
19	1015	True	True	False	True	True
20	1016	True	True	False	True	True
21	1017	True	True	False	True	False
22	1018	True	False	False	True	True
23	1019	True	True	False	True	False
24	102	True	True	False	True	False
25	1020	True	False	False	True	True
26	1021	True	False	False	True	True
27	1022	True	True	False	True	False
28	1023	True	False	False	True	False
29	1024	True	True	False	True	False
...	...	...	...	...	...	...
1122	972	True	False	False	False	True
1123	973	True	True	False	True	True
1124	974	True	True	False	False	True
1125	975	True	True	False	False	True
1126	976	True	False	False	True	False
1127	977	True	True	False	False	True
1128	978	True	False	False	True	False
1129	979	True	False	False	False	True
1130	98	True	True	False	False	False
1131	980	True	True	False	True	True
1132	981	True	False	False	False	False
1133	982	True	False	False	True	True
1134	983	True	False	False	True	True
1135	984	True	True	False	True	True
1136	985	True	False	False	False	True
1137	986	True	False	False	False	False
1138	987	True	False	False	True	True
1139	988	True	False	False	True	True
1140	989	True	False	False	False	True
1141	99	True	False	False	True	True
1142	990	True	True	False	False	False
1143	991	True	False	False	False	True
1144	992	True	True	False	False	False
1145	993	True	False	False	True	False
1146	994	True	True	False	False	False
1147	995	True	True	False	True	True
1148	996	True	True	False	True	False
1149	997	True	False	False	False	False
1150	998	True	True	False	True	False
1151	999	True	False	False	True	True

	no_fast_pskip	no_mbtree	no_mixed_refs	no_weightb	rc_lookahead	ref \
0	True	False	True	True	20	9
1	True	False	False	True	40	9
2	False	True	True	False	40	1
3	True	True	True	False	40	9
4	False	False	True	False	60	5
5	True	False	False	False	60	5
6	False	False	True	False	60	1
7	True	False	False	False	60	1
8	True	False	False	True	60	1
9	True	False	False	True	60	9
10	True	False	False	True	60	5
11	False	False	True	False	60	9
12	False	False	False	True	20	1
13	False	True	False	False	40	5
14	True	False	False	False	60	9
15	True	False	False	False	20	1
16	True	True	False	False	60	5
17	True	True	True	False	40	1
18	True	True	False	False	60	1
19	True	False	False	False	20	9
20	True	False	False	False	20	5
21	False	False	True	False	60	1
22	True	True	True	False	40	9
23	True	True	True	False	60	1
24	True	True	False	False	40	1
25	True	True	True	False	40	5
26	True	True	False	False	60	9
27	True	True	True	False	60	5
28	True	False	True	False	60	1
29	True	True	True	False	60	9
...	...	...	...	...	...	...
1122	False	False	True	False	40	1
1123	False	True	False	False	20	1
1124	True	False	True	False	20	5
1125	True	False	True	False	20	1
1126	True	False	False	False	40	9
1127	True	False	True	False	20	9
1128	True	False	False	False	40	5
1129	False	False	True	False	40	9
1130	True	True	True	True	40	1
1131	False	True	True	True	20	9
1132	False	True	True	False	40	9
1133	True	True	True	False	60	5
1134	False	False	False	True	20	5
1135	False	True	True	True	20	5
1136	True	True	True	True	40	9
1137	False	True	True	False	40	5

1138	True	True	True	False	60	1
1139	False	False	False	True	20	9
1140	True	True	True	True	40	5
1141	True	False	True	False	60	9
1142	True	False	True	True	40	5
1143	True	True	True	True	40	1
1144	True	False	True	True	40	9
1145	True	False	True	True	60	9
1146	True	False	True	True	40	1
1147	False	True	True	True	20	1
1148	True	True	True	False	40	1
1149	False	True	True	False	40	1
1150	True	True	True	False	40	5
1151	True	True	True	False	60	9

	size	usertime	systemtime	elapsedtime
0	14580447	64.8645	0.4290	0.719094
1	16245143	77.1640	0.5080	0.932402
2	8470682	37.3405	0.2970	0.396907
3	9136214	62.1315	0.4235	0.674512
4	16614803	53.3095	0.5360	0.658043
5	17356184	59.8465	0.5165	0.772645
6	17021341	44.7380	0.4560	0.542240
7	17718070	45.4100	0.4635	0.554206
8	15879470	42.1435	0.4245	0.513267
9	15351974	77.4560	0.5325	0.943058
10	15462849	56.8100	0.4910	0.704078
11	16498606	67.9350	0.5165	0.795465
12	16905997	44.4760	0.3905	0.498789
13	8976078	53.4425	0.3880	0.617248
14	17215198	79.8385	0.5185	0.997919
15	15975804	42.1550	0.3775	0.470264
16	9622163	53.3200	0.4240	0.622576
17	9723870	39.0725	0.3110	0.420606
18	9723870	39.0495	0.3285	0.420821
19	15564911	76.2745	0.4780	0.901455
20	15685570	56.4345	0.4760	0.676683
21	16794642	43.2630	0.4655	0.534544
22	9656148	63.7380	0.4260	0.693852
23	9182452	37.2595	0.3075	0.404998
24	9182452	37.2415	0.2995	0.404855
25	9675772	48.4975	0.3945	0.558350
26	9598511	73.2260	0.4440	0.835274
27	9156301	46.7880	0.3820	0.532319
28	17718070	45.4990	0.4330	0.558583
29	9136214	62.1555	0.3895	0.674064
...	...	...	...	...
1122	16834417	44.3300	0.4215	0.513069

```
1123   9248498  37.2135      0.3040     0.398576
1124  14635822  50.0745      0.4230     0.567374
1125  14880609  41.2570      0.3650     0.454280
1126  17053063  79.2790      0.5075     0.965214
1127  14559814  64.8915      0.4325     0.711451
1128  17190573  59.4410      0.4780     0.744730
1129  16326834  67.6255      0.4585     0.763083
1130   8409042  37.4445      0.3100     0.397338
1131   9165923  61.3020      0.3880     0.667426
1132   9012056  63.6520      0.3900     0.688470
1133   9675772  48.5145      0.3960     0.555695
1134  16536068  58.8030      0.4555     0.710949
1135   9185491  46.3095      0.3830     0.523421
1136   9026749  64.2035      0.4010     0.694444
1137   9020689  48.7605      0.4080     0.548519
1138   9723870  39.1240      0.3160     0.421772
1139  16398226  78.8260      0.4780     0.938340
1140   9033806  48.8485      0.4135     0.543855
1141  17350571  68.4065      0.4940     0.806014
1142  15312305  50.8550      0.4500     0.600456
1143   9070760  39.1985      0.3180     0.415403
1144  15225140  65.6965      0.4890     0.741878
1145  17381051  68.9510      0.5240     0.807395
1146  15617786  42.0910      0.4175     0.486805
1147   9238655  37.2395      0.3100     0.398522
1148   9182452  37.1980      0.3235     0.401554
1149   9033164  39.2840      0.3035     0.415421
1150   9156301  46.7860      0.3705     0.531117
1151   9656148  63.7225      0.4340     0.693762

[1152 rows x 16 columns]
```

In [32]:
```
a = 1.23
```

1.1 1.23

1.23

In [33]:
```
import notebook
#notebook.install_nbextension('python-markdown',user=True)
```

In [34]:
```
E=notebook.nbextensions.EnableNBExtensionApp()
```

In [35]:
```
E.print_version()
```

```
5.7.2
```

In [36]:
```
notebook.nbextensions.check_nbextension('python-markdown', user=True)
```

```
Out[36]: True
```

```
In [ ]:
```