Are different adverse events reported in different countries?

The problem: adverse events are reported world wide to the FDA. Based on these reported events, additional studies and investigations are required in the drug life cicle. However, are these events are global or it is country specific?

The purpose of this task is to investigate if there is differences in the events reported between different countires. The idea is to check if there is a similarity between the countries accorind to the reported events. The events are associated with certain drug, and building the system as drug independent would make it less accurate. For that purpose, the analysis is drug dependent. However, for observation purposes if some countires are only reported for serious events only, we present the result of drug independent analysis.

The hypothesis: the null hypothesis of this study that there is no differences in the reported events per country.

The alternative hypothesis

Hypothesis 1 : each country is significantly differnt to at least another country regards the reported adverse events.

Hypothesis 2 : each country is significantly differnt to all other countries regards the reported adverse events.

To discover such similarity relationship in the data. We calculate the significant disagreement between each pairs of countries regards to the events reported for each of them. Such analysis would validate the proposed hypothesis. For deeper analysis, we are going to use the collaborative filtering as a mechanism to decompose the relationsship between the countries using the reported events. The collaborative filtering is a techniques wildely used in recommendation systems in order to the obtaining recommendations from the similarities. The idea based on the fact that similar items have similar history. To project that on the proposed question, the more similar the counties are the more common events they have and the more they are differnt, the less common events they have. Such task requires the following steps:

1- data collection.

2- data curation.

3- data analysis

4- model building and result analysis and visualization.

In [107]:
```
import requests
import json
from datapackage import Package
import pandas as pd
import numpy as np
from scipy.sparse import csc_matrix
from scipy import stats
from sklearn.manifold import TSNE
from bokeh.plotting import figure, show
from bokeh.models import ColumnDataSource, Range1d, LabelSet, Label
import bokeh.plotting as bkp
import bokeh.models as bkm
from bokeh.io import output_notebook
from bokeh.models import LogColorMapper
from bokeh.palettes import Viridis6 as palette
from bokeh.plotting import figure
from bokeh.layouts import row
import implicit
from sklearn.decomposition import PCA
from implicit.als import AlternatingLeastSquares
from implicit.approximate_als import (AnnoyAlternatingLeastSquares, FaissAlternatingLeastSquares,
                                      NMSLibAlternatingLeastSquares)
from implicit.bpr import BayesianPersonalizedRanking
from implicit.nearest_neighbours import (BM25Recommender, CosineRecommender,
                                         TFIDFRecommender, bm25_weight)
output_notebook()
```

(https://bokeh.pydata.org) BokehJS 0.13.0 successfully loaded.

Data collecting:

In order to perform the analysis, we need to collect the data needed. The data is stored in FDA server and could be collected using the event API. The event API is searchable on multi fields and count on one only. A generic function is built that takes the searchable fields, the count field and the number of results to return, and return a json format results

In [108]:
```
def execute_fda_query(search_conditions=None,count_condition=None,limit='1'):
    '''
    this function create a fda api request and return the result in json format
    Params:
        search_conditions: list of the searchable conditions (ex. ['patient.drug.openfda.generic_name.exact:"ASPIRIN"'])
        count_condition: a string the count condition (ex.patient.reaction.reactionmeddrapt.exact)
        limit: a string the number of result size (due to API constraints, the result can't exceed 1000 for count and
        100 per page for the patient data)
    Returns:
        string
    '''
    END_POINT_SERVER="https://api.fda.gov/drug/event.json?"
    if(search_conditions!=None):
        search_string='search='
        search_string+='+AND+'.join(search_conditions)
        END_POINT_SERVER+=search_string+"&"
    if(count_condition!=None):
        count_condition='count={}'.format(count_condition)
        END_POINT_SERVER+=count_condition+"&"
    limit_string='limit={}'.format(limit)
    END_POINT_SERVER+=limit_string
    response = requests.get(END_POINT_SERVER)
    return response.json()
```

In this task, we will perform the analysis for the top drugs those regards the number of reports recieved. we use the generic medical name of the durgs.

In [109]:
```
def get_top_generic_meds(limit):
    '''
    get the top 'limit' drug generic names reported have the highest adverse ev
ents reported
    Params:
        limit: the number of results to return
    Returns:
        list
    '''
    search_conditions=None
    count_condition='patient.drug.openfda.generic_name.exact'
    result=execute_fda_query(count_condition=count_condition,limit=limit)
    return result['results']
```

After having these durg names, we need to have the countries. The countries are stored in the API by their code, for that we will retrieve these codes from the data package library and return a list of the country codes.

In [110]:
```
countries_info=pd.read_csv('countries.csv')
def get_countries():
    '''
    get the countries alpha-2 codes
    Params:
        limit: the number of results to return
    Returns:
        list
    '''
    return countries_info['alpha-2'].tolist()
```

Once we have the county codes and the durg ids, we can retrieve the events associated with this durg in that counry using the FDA API.

In [111]:
```
def get_events_by_county(med,country,limit):
    '''
    get the top 'limit' adverse events with their reported counts for certain c
ountry and certain drug
    Params:
        med: the generic drug name
        country: the country alpha-2 code
        limit: the number of results to return
    Returns:
        list
    '''
    search_conditions=[]
    search_conditions.append('patient.drug.openfda.generic_name.exact:"'+med+'"
')
    search_conditions.append('occurcountry:"'+country+'"')
    count_condition='patient.reaction.reactionmeddrapt.exact'
    result=execute_fda_query(search_conditions,count_condition,limit=limit)
    return result

def get_full_drugs_events_by_county(country,limit):
    '''
    get the top 'limit' adverse events with their reported counts for certain c
ountry
    Params:
        country: the country alpha-2 code
        limit: the number of results to return
    Returns:
        list
    '''
    search_conditions=[]
    search_conditions.append('occurcountry:"'+country+'"')
    count_condition='patient.reaction.reactionmeddrapt.exact'
    result=execute_fda_query(search_conditions,count_condition,limit=limit)
    return result
```

For writing the output list into file, we have the helper function to do so

Data curation and saving

Several data curation steps could be performed, such as: Wrong adverse event enties (AND, MG, OFF LABEL USE ..) Synonyms Unclear event name (DRUG INTERACTION) This type of curation requires a time and deeper analysis or a domain knowledge. For that purpose, an example of what we assumed as wrong adverse event enties and unclear event name will be removed. A more safisticated data curation is required

In [112]:
```
excluded_events=['DRUG INTERACTION','OFF LABEL USE','MG','AND']
def remove_event(eventname):
    return eventname in excluded_events
```

In [113]:
```
def save_country_data(path,number_of_meds):
    '''
    this function retrieve the top 'N' drug name and save the reported event co
unts per country for each drug seperately
    (i.e. drug specific data)
    Params:
        path: the name of the output folder to save the data in (the filename f
or each drug would be [DRUG_NAME]_events.csv).
        number_of_meds: the number of top drugs to save their data
    Returns:
        None
    '''
    top_generic_meds=get_top_generic_meds(number_of_meds)
    countries=get_countries()
    for med in top_generic_meds :
        print 'retreiving {} data'.format(str(med['term']))
        outputlist=[]
        outputlist.append('country,term,count')
        for country in countries:
            top_events=get_events_by_county(med['term'],country,'1000')
            if('results' in top_events):
                top_events=top_events['results']
                for elm in top_events:
                    if(not remove_event(str(elm['term']).replace(",","_"))):
                        outputlist.append(str(country).replace(",","_")+","+str
(elm['term']).replace(",","_")+','+str(elm['count']))
        filename=path+med['term']+'_events.csv'
        write_to_file(outputlist,filename)

def save_full_country_data(path):
    '''
    this function retrieve save the reported event counts per country for all d
rugs
    Params:
        path: the name of the output folder to save the data in  (the filename
would be full_events.csv).
    Returns:
        None
    '''
    outputlist=[]
    countries=get_countries()
    for country in countries:
        top_events=get_full_drugs_events_by_county(country,'1000')
        if('results' in top_events):
            top_events=top_events['results']
            for elm in top_events:
                if(not remove_event(str(elm['term']).replace(",","_"))):
                    outputlist.append(str(country).replace(",","_")+","+str(elm
['term']).replace(",","_")+','+str(elm['count']))
    filename=path+'full_events.csv'
    write_to_file(outputlist,filename)
```

In [114]:
```
# save_full_country_data('./data/')
# save_country_data('./data/',20)
```

Model building:

Now we have the data retieved, curated and saved on the storage, we can start do the analysis and building building the model. In order to do so we need to read the data filter it if needed and convert it to the right format.

For that purpose, it works on a matrix represents the country/event relationship. The number of rows is the number of countries and the number of columns is the number of events. The entry in the country/event cell is the number of reports mention the event and the report source is row country.

First we read the data file content and filter the content according to the number of top events to consider. If the number of events to retrieve variable is 0, we return all the events. This parameter is controling the spasity of the matrix, the bigger the number is the less sparse the matrix is. However, that might give a consideration for rare events.

```
In [115]: countries_info=pd.read_csv('countries.csv')
```

```
In [116]: def get_top_events(file_path,number_of_events_to_retrieve):
              '''
              This function read a adverse events per country data and return it in dataf
          rame structure.
              If the number of events to retrieve is 0, we return all data. If not, we re
          turn the top number_of_events_to_retrieve
              of events per country
              Params:
                  file_path: the full input file path
                  number_of_events_to_retrieve: the number of top adverse event per count
          ry to retrieve.
              Returns:
                  DataFrame
              '''
              df = pd.read_csv(file_path)
              if(number_of_events_to_retrieve==0):
                  return df
              result=[]
              countries=df.country.unique().tolist()
              for elm in countries:
                  a=df[df['country']==elm]
                  res=a.head(number_of_events_to_retrieve)
                  result.append(res)
              return pd.concat(result)
```

```
In [117]: def delete_by_threshold(df,threshold):
              dfx=df[df['count']> threshold]
              return dfx
```

Now, we put the content of the file into a sparse matrix. We have Three types of data to include in this matrix, the first one is the actual number of reports for that event in the country and the other only the existance of a report. The first one would work on the actual count and that would be part of the modeling. However, due to the difference in the number of reports per country (health care problems, only serious reporting, etc..), the differences in the number would be projected as difference in weights inside the model (some models would ignore the values as cosine distance). The second is the ratio to the total number of events of that country. The third would be a binary if the country has such event report or not. The other parameter is the threshold, which decide the minimum number of reported events in order to consider that event as part of the country data.

In [118]:
```
def convert_to_ratio(df):
    '''
    Convert the data in the df into ratio data, that's by taking the count of e
ach event for each country and divide
    it on the sum of the reported events for that country.
    Params:
        df: the data dataframe
    Returns:
        DataFrame
    '''
    result=[]
    countries=df.country.unique().tolist()
    for elm in countries:
        a=df[df['country']==elm]
        a['count']=a['count']/a['count'].sum()
        result.append(a)
    return pd.concat(result)
```

In [119]:
```
def file_to_sparse_matrix(file_path,number_of_events_to_show=0,threshold=0,norm
alized=False,ratio=False):
    '''
    This function takes the name of the input file, with several configration p
arameters and returns
    the countr/event matrix, a list of the code of the countries in the data, l
ist of the events mentioned
    and the original data in dataframe format.
    Params:
        file_path:
        number_of_events_to_show: the maximum number of top events per country
to read.
        threshold: the minimum number of report per event to be considered.
        normalized: convert the data into binary ( instead of count, just menti
on event reported/non-reported)
        ratio: convert the event counts into ratio format per country.
    Returns:
        SparseMatrix, list, list, DataFrame
    '''
    df=get_top_events(file_path,number_of_events_to_show)
    if(threshold>0):
        df=delete_by_threshold(df,threshold)
    countries=df.country.unique().tolist()
    countries_index=[]
    events_index=[]
    if(ratio):
        df=convert_to_ratio(df)
    events=df.term.unique().tolist()
    values=df['count'].tolist()
    if(normalized):
        values=df['count']*0+1
    for elm in df.country:
        countries_index.append(countries.index(elm))
    for elm in df.term:
        events_index.append(events.index(elm))
    sparse_mat=csc_matrix((values, (countries_index, events_index)), shape=(len
(countries), len(events)))
    return sparse_mat,countries,events,df
```

Now, as the data is in the right format, we can build the collaborative model. There is several types of the collaborative models to use: Item-to-item Nearest Neighbour Based: this approach is based on the distance similarity between the items, such as the cosine distance or Term Frequency and Inverse Document Frequency meausre. Matrix Decomposition Based: in this class of CF, the goal it to uncover latent features that explain observed values. Several approaches have been proposed for such case as - Alternating Least Squares -Bayesian Personalized Ranking. Unlike the matrix decomposition which require the tuning of the number of factors, the item-to-item approach does not require any parameters.

In [120]:
```
MODELS = {"als":  AlternatingLeastSquares,
          "tfidf": TFIDFRecommender,
          "cosine": CosineRecommender,
          "bpr": BayesianPersonalizedRanking}
def get_model(model_name,num_of_factors):
    '''
    create a collaborative filtering model by name.
    Params:
        name: the name of the model to generate
        num_of_factors: the number of latent factor if the model is matrix deco
mposition based.
    Returns:
        Model
    '''
    model_class = MODELS.get(model_name)
    if not model_class:
        raise ValueError("Unknown Model '%s'" % model_name)
    if issubclass(model_class, AlternatingLeastSquares):
        params = {'factors': num_of_factors, 'dtype': np.float32}
    elif model_name == "bpr":
        params = {'factors': num_of_factors}
    else:
        params = {}
    return model_class(**params)
```

Now we have the model creation code, we can create a model and train it on the data matrix.

In [121]:
```
def create_fit_CF_model(modelname,item_user_data,num_of_factors):
    '''
    this function takes the model name, the country/event_matrix and the number
of factors and return a CF model trained
    on the input data.
    Params:
        modelname: the name of the model to generate
        item_user_data: the data to train the model on.
        num_of_factors: the number of latent factor if the model is matrix deco
mposition based.
    Returns:
        Model
    '''
    model =get_model(modelname,num_of_factors)
    model.fit(item_user_data)
    return model
```

In [122]:
```
def get_country_matrix_similarity(model,countries):
    '''
    this function takes the model and a set of counry, and it return the simila
rity scores to all other countries
    as a matrix.
    Params:
        model: the CF model
        countries: the list of countries to generate the similarities to.
    Returns:
        list of lists
    '''
    result=[]
    for i in range(len(countries)):
        scores=[0]*len(countries)
        for value in model.similar_items(i, len(countries)):
            scores[value[0]]=value[1]
        result.append(scores)
    return result
```

In [123]:
```
def show_country_similarity(score_matrix,countries):
    '''
    this function takes a similarity score matrix and the code of the countrie,
project the similarity as a distance
    between the countries using t-sne and plot it into 2D graph as scatter.
    Params:
        score_matrix: the similarity score matrix
        countries: the country codes
    Returns:
        None
    '''
    score_matrix=(score_matrix-score_matrix.min())/(score_matrix.max()-score_ma
trix.min())
    X_embedded=TSNE(n_components=2).fit_transform(score_matrix)
    countriesnames=[]
    report_count=[]
    number_of_events_to_show=3
    report_text={}
    colormap = ["#444444", "#1f78b4", "#33a02c", "#fb9a99",
                "#e31a1c", "#ff7f00", "#cab2d6", "#6a3d9a"]
    continents=countries_info['region'].unique().tolist()
    colors=[]
    for i in range(number_of_events_to_show):
        report_text[i]=[]
    label=[]
    for elm in countries:
        label.append(countries_info[countries_info['alpha-2']==elm]['region'].t
olist()[0])
        colors.append(colormap[continents.index(countries_info[countries_info['
alpha-2']==elm]['region'].tolist()[0])])
        countriesnames.append(countries_info[countries_info['alpha-2']==elm]['n
ame'].values.tolist()[0])
        a=df[df['country']==elm]
        res=a.head(number_of_events_to_show)['term'].tolist()
        full_score=a['count'].sum()
        report_count.append(full_score)
        scores=a.head(number_of_events_to_show)['count'].tolist()
        for i in range(number_of_events_to_show):
            if(i<len(res)):
                report_text[i].append(res[i]+' : '+str(scores[i])+'/'+str(full_
score))
            else:
                report_text[i].append('')
    report_count=(np.round(report_count/(np.max(report_count)/50.0))+5).tolist(
)
    source = ColumnDataSource(data=dict(height=X_embedded[:,0].tolist(),
                                    weight=X_embedded[:,1].tolist(),
                                    names=countriesnames,
                                    codes=countries,
                                    report_count=report_count,
                                    report_text1=report_text[0],
                                    report_text2=report_text[1],
                                    report_text3=report_text[2],
                                    colors=colors,
                                    label=label))
    p = figure(title='countries',plot_width=800, plot_height=500,tooltips=[("Na
me", "@names")
                                                                            ,("Fir
st Event", "@report_text1")
                                                                            ,("Sec
ond Event", "@report_text2")
                                                                            ,("Thi
rd Event", "@report_text3")])
    p.scatter(x='height',y='weight', size=5, source=source,legend='label',color
='colors')
    p.xaxis[0].axis_label = 'Comp1'
    p.yaxis[0].axis_label = 'Comp2'
    show(p)
```

In [124]:
```
def get_outliers(input_matrix,countries)  :
    '''
    this function takes a similarity score matrix and the code of the countrie
and return a set of statistics about
    the data. These statistics (per country) are median,mean,max,min,std,outlie
rs
    Params:
        input_matrix: the similarity score matrix
        countries: the country codes
    Returns:
        list,list,list,list, list of lists
    '''
    yy = []
    g = []
    for i in range(input_matrix.shape[0]):
        for j in range(input_matrix.shape[1]):
            g.append(countries[i])
            if i!=j:
                yy.append(input_matrix[i,j])
            else:
                yy.append(0)
    yy=np.asarray(yy)
    g=np.asarray(g)
    dfx = pd.DataFrame(dict(score=yy, group=g))
    # find the quartiles and IQR for each category
    groups = dfx.groupby('group')
    q1 = groups.quantile(q=0.25)
    q2 = groups.quantile(q=0.5)
    q3 = groups.quantile(q=0.75)
    iqr = q3 - q1
    upper = q3 + 1.5*iqr
    lower = q1 - 1.5*iqr
    # find the outliers for each category
    def outliers(group):
        cat = group.name
        return len(group[(group.score > upper.loc[cat]['score'])])
    def outliers_counts(group):
        cat = group.name
        return group[(group.score > upper.loc[cat]['score']) | (group.score < l
ower.loc[cat]['score'])]
    out = groups.apply(outliers).dropna()
    return q2,groups.mean(),groups.max(),groups.min(),groups.std(),out
```

In [125]:
```
def plot_box(input_matrix,labels,width,hight):
    '''
    this function takes an input matrix, labels and the width and hights and sh
ow the plot box of the data.
    Params:
        input_matrix: the similarity score matrix
        labels: the row labels
        width: the plot width
        hight: the plot hight
    Returns:
        None
    '''
    yy = []
    g = []
    for i in range(input_matrix.shape[0]):
        for j in range(input_matrix.shape[1]):
            g.append(labels[i])
            if i!=j:
                yy.append(input_matrix[i,j])
            else:
                yy.append(0)
    yy=np.asarray(yy)
    g=np.asarray(g)
    dfx = pd.DataFrame(dict(score=yy, group=g))
    # find the quartiles and IQR for each category
    groups = dfx.groupby('group')
    q1 = groups.quantile(q=0.25)
    q2 = groups.quantile(q=0.5)
    q3 = groups.quantile(q=0.75)
    iqr = q3 - q1
    upper = q3 + 1.5*iqr
    lower = q1 - 1.5*iqr
    # find the outliers for each category
    def outliers(group):
        cat = group.name
        return group[(group.score > upper.loc[cat]['score']) | (group.score < l
ower.loc[cat]['score'])]['score']
    out = groups.apply(outliers).dropna()
    # prepare outlier data for plotting, we need coordinates for every outlier.
    if not out.empty:
        outx = []
        outy = []
        for cat in labels:
            # only add outlliers if they exist
            try:
                if not out.loc[cat].empty:
                    for value in out[cat]:
                        outx.append(cat)
                        outy.append(value)
            except:
                continue
    p = figure(background_fill_color="#EFE8E2",plot_width=width, plot_height=hi
ght, title="", x_range=labels)
    # if no outliers, shrink lengths of stems to be no longer than the minimums
or maximums
    qmin = groups.quantile(q=0.00)
    qmax = groups.quantile(q=1.00)
    upper.score = [min([x,y]) for (x,y) in zip(list(qmax.loc[:,'score']),upper.
score)]
    lower.score = [max([x,y]) for (x,y) in zip(list(qmin.loc[:,'score']),lower.
score)]
    # stems
    p.segment(labels, upper.score, labels, q3.score, line_color="black")
    p.segment(labels, lower.score, labels, q1.score, line_color="black")
    # boxes
    p.vbar(labels, 0.7, q2.score, q3.score, fill_color="#E08E79", line_color="b
lack")
    p.vbar(labels, 0.7, q1.score, q2.score, fill_color="#3B8686", line_color="b
lack")
```

3- Data Analysis

Now we start with first one, which is checking if there is a significant difference between the events reported for each pair of countries. As we are working with correlation problem between ordinal features, Sign test is used to calculate the degree of agreement on the events between two counties. The sign test has the null hypothesis that both samples are from the same population. The sign test compares the two dependent observations and counts the number of negative and positive differences. First we load th data used in the experiment. As we do not consider the count of events as a factor here, we should care about if reported or not, so re binarize the data accordenly

In [126]:
```
# The input file
file_name='full_events.csv'
# The minimum number of reports to consider per event
threshold=0
#  the maximum number of events to consider per country
number_of_events_per_country=1000
# use the raw data or a ratio per country
ratio=False
# use binary or float (True means there is event reported in that country witho
ut how many)
normalized=True
# create the dataset
sparse_mat,countries,events,df=file_to_sparse_matrix(file_name,number_of_events
_to_show=number_of_events_per_country,threshold=threshold,normalized=normalized
,ratio=ratio)
a=sparse_mat.toarray()
```

Now, we examine the null hypothesis, that all countries have same adverse events. That means, if we found countries those are significantly different regards to the events reported, we can reject the null hypothsis. We are interested in the count of the significant (p-value<0.05) with disagreement >0. The disagreement is calculated and the sum of the absolute difference between the event occurance between the country i and country j. The significant check is done using the sign test for the difference between the event occurance between the country i and country j.

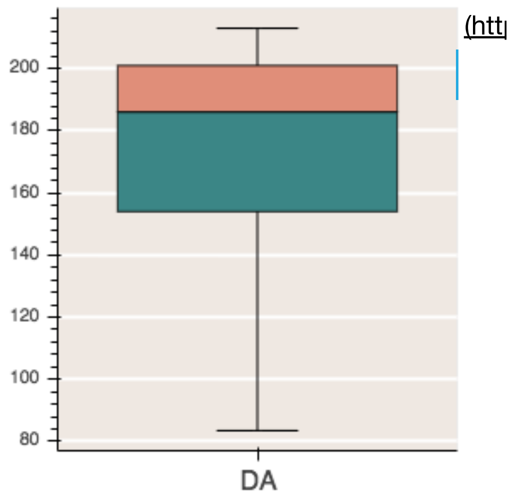
In [127]:
```
def sign_test(samp, mu0=0):
    '''
    taken from
    https://github.com/statsmodels/statsmodels/blob/master/statsmodels/stats/de
scriptivestats.py
    '''
    samp = np.asarray(samp)
    pos = np.sum(samp > mu0)
    neg = np.sum(samp < mu0)
    M = (pos-neg)/2.
    p = stats.binom_test(min(pos,neg), pos+neg, .5)
    return M, p
```

In [128]:

```
seg_diff_count=[]
non_seg_diff_count=[]
diff_count=[]
perfect_sig_count=[]
for j in range(a.shape[0]):
    res=0
    perfect_sig=0
    non_seg_res=0
    for i in range(a.shape[0]):
        if(i==j):
            continue
        difference=np.sum(np.abs(a[i,:]-a[j,:]))*1.0
        M,p=sign_test(a[i,:]-a[j,:])
        if(p<0.05):
            if difference!=0:
                res+=1
                diff_count.append(difference/len(a[i,:]))
            else:
                perfect_sig+=1
        else:
            if difference==0:
                perfect_sig+=1
    if(res>0):
        seg_diff_count.append(res)
    if(perfect_sig>0):
        perfect_sig_count.append(perfect_sig)
seg_diff_count=np.asarray(seg_diff_count)
perfect_sig_count=np.asarray(perfect_sig_count)
diff_count=np.asarray(diff_count)
```

In [129]:
```
print "Segnificient disagreement per country : {} out of {}, with minimun {} co
unties difference and maximum of {} ".format(int(seg_diff_count.mean()),a.shape
[0],seg_diff_count.min(),seg_diff_count.max())
print "Average significant disagreement : {}% ".format(100-(diff_count.mean()*1
00))
print "{} out of {} countries have at least {} countries have significant diffe
rent adverse events".format(len(seg_diff_count),a.shape[0],seg_diff_count.min()
)
plot_box(np.asarray([seg_diff_count]),['DA'],300,300)
if(len(perfect_sig_count)==0):
    print "The number of countries those show full agreement : 0 out of {} ".fo
rmat(a.shape[0])
else:
    print "The number of countries those show full agreement: {} out of {} ".fo
rmat(len(perfect_sig_count),a.shape[0])
```

```
Segnificient disagreement per country : 182 out of 214, with minimun 153 count
ies difference and maximum of 213
Average significant disagreement : 90.5656694794%
214 out of 214 countries have at least 153 countries have significant differen
t adverse events
```



```
The number of countries those show full agreement: 3 out of 214
```

As shown in the results, each coutrie is significantly different from at least 153 other countires. In average, each country is significantly different from 186 other countries regards the reported events. The average disagreement degree is 90%. Figure DA shows the box plot of the number of significant different country count. Only Three countries those have a full match with others. That's to say, the null hypothsis that all countries show same events can be rejected.

Now for the Hypothesis 1 : each country is significantly differnt to at least another country regards the reported adverse events.

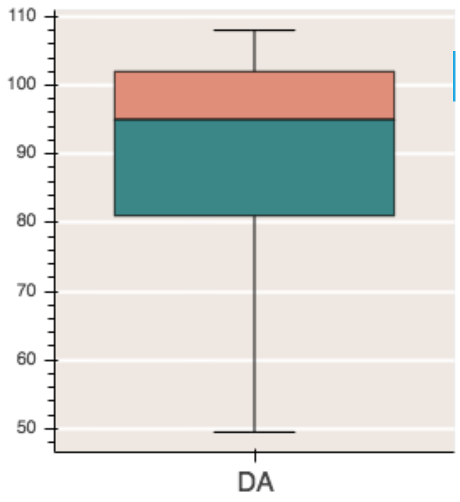
From the result, we notice that all the countries have at least one another country that is significantlly disagree about the reported event. Which make this hypothesis a valid one.

Hypothesis 2 : each country is significantly differnt to all other countries regards the reported adverse events. The results show that there is 3 countries those have some fully agreement with other countries. Which make the hypothesis 2 not valid.

Now if we repeat this analysis for a drug specific case:

In [130]:
```
# The input file
file_name='./data/CLOPIDOGREL BISULFATE_events.csv'
# The minimum number of reports to consider per event
threshold=0
#  the maximum number of events to consider per country
number_of_events_per_country=1000
# use the raw data or a ratio per country
ratio=False
# use binary or float (True means there is event reported in that country witho
ut how many)
normalized=True
# create the dataset
sparse_mat,countries,events,df=file_to_sparse_matrix(file_name,number_of_events
_to_show=number_of_events_per_country,threshold=threshold,normalized=normalized
,ratio=ratio)
a=sparse_mat.toarray()
seg_diff_count=[]
non_seg_diff_count=[]
diff_count=[]
perfect_sig_count=[]
for j in range(a.shape[0]):
    res=0
    perfect_sig=0
    non_seg_res=0
    for i in range(a.shape[0]):
        if(i==j):
            continue
        difference=np.sum(np.abs(a[i,:]-a[j,:]))*1.0
        M,p=sign_test(a[i,:]-a[j,:])
        if(p<0.05):
            if difference!=0:
                res+=1
                diff_count.append(difference/len(a[i,:]))
            else:
                perfect_sig+=1
        else:
            if difference==0:
                perfect_sig+=1
    if(res>0):
        seg_diff_count.append(res)
    if(perfect_sig>0):
        perfect_sig_count.append(perfect_sig)
seg_diff_count=np.asarray(seg_diff_count)
perfect_sig_count=np.asarray(perfect_sig_count)
diff_count=np.asarray(diff_count)
print "Segnificient disagreement per country : {} out of {}, with minimun {} co
unties difference and maximum of {} ".format(int(seg_diff_count.mean()),a.shape
[0],seg_diff_count.min(),seg_diff_count.max())
print "Average significant disagreement : {}% ".format(100-(diff_count.mean()*1
00))
print "{} out of {} countries have at least {} countries have significant diffe
rent adverse events".format(len(seg_diff_count),a.shape[0],seg_diff_count.min()
)
plot_box(np.asarray([seg_diff_count]),['DA'],300,300)
if(len(perfect_sig_count)==0):
    print "The number of countries those show full agreement : 0 out of {} ".fo
rmat(a.shape[0])
else:
    print "The number of countries those show full agreement: {} out of {} ".fo
rmat(len(perfect_sig_count),a.shape[0])
```

```
Segnificient disagreement per country : 91 out of 109, with minimun 74 countie
s difference and maximum of 108
Average significant disagreement : 90.8950617284%
109 out of 109 countries have at least 74 countries have significant different
adverse events
```



(htt

```
The number of countries those show full agreement: 2 out of 109
```

As we can see, the drug specific case shows very similar results to the generic one.

Next section we do a deeper analysis using the collaborative filtering (CF)

4- Model Building

In [131]:
```
# The input file
file_name='full_events.csv'
# number of factors for the MD approaches
num_of_factors=300
# use the raw data or a ratio per country
ratio=True
# use binary or float (True means there is event reported in that country witho
ut how many)
normalized=False
# The minimum number of reports to consider per event
threshold=3
#  the maximum number of events to consider per country
number_of_events_per_country=200
# create the dataset
sparse_mat,countries,events,df=file_to_sparse_matrix(file_name,number_of_events
_per_country,threshold,normalized,ratio)
# the CF approaches to compare
models_to_examine=["als","bpr","cosine","tfidf"]
models={}
score_matrices={}
for modelname in models_to_examine:
    model=create_fit_CF_model(modelname,sparse_mat,num_of_factors)
    models[modelname]=model
    score_matrix=get_country_matrix_similarity(models[modelname],countries)
    score_matrices[modelname]=np.asarray(score_matrix)
```

```
/Users/mohamedabou-zleikha/venv/lib/python2.7/site-packages/ipykernel_launcher
.py:14: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/sta
ble/indexing.html#indexing-view-versus-copy

100%|██████████| 15.0/15 [00:01<00:00, 12.94it/s]
100%|██████████| 100/100 [00:01<00:00, 60.10it/s, skipped=50.84%, correct=81.5
6%]
100%|██████████| 159/159 [00:00<00:00, 23989.00it/s]
100%|██████████| 159/159 [00:00<00:00, 19942.42it/s]
```

In [132]:
```
# to check the existances of differences or similarity, we could analysis the s
imilarity values.
# for cosine based similarity, the values are between 0 and 1, and the closer t
he value to 1,
# the higher the similarity is. The result are the average over all countries.
for modelname in score_matrices:
    score_matrix1=np.asarray(score_matrices[modelname])
    score_matrix1=(score_matrix1-score_matrix1.min())/(score_matrix1.max()-scor
e_matrix1.min())
    medV,meanV,maxV,minV,stdV,mean_num_outliers=get_outliers(score_matrix1,coun
tries)
    print modelname
    print 'Med : {}, Std : {}, Max : {}, Min: {},Avg Outliers: {}'.format(medV.
score.mean(),stdV.score.mean(),maxV.score.mean(),minV.score.mean(),mean_num_out
liers[0])
```

```
tfidf
Med : 0.0, Std : 0.0793487862529, Max : 0.391175451154, Min: 0.0,Avg Outliers:
19
cosine
Med : 0.0, Std : 0.206276193391, Max : 0.723955627954, Min: 0.0,Avg Outliers:
19
bpr
Med : 0.551427679242, Std : 0.239352901213, Max : 0.963106887146, Min: 0.0,Avg
Outliers: 0
als
Med : 0.237906594415, Std : 0.102022835908, Max : 0.554257897648, Min: 0.0,Avg
Outliers: 1
```

As we can see, the average similaities in the item-to-item approaces are very low with a Med value and the std is 0.07 and 0.2 , which means we have 95% of the country below the 0.5 similarity. We can also notice that average of the positive outleiers (which are the candidates to be similar) are 19.

For the MD based similarities, the average is higher and the avarage number of outliers is lower.This is due to the fact that the matrix decomposition would descover deeper relations using the latent comparing with the relations extracted by the cosine similarity.

In this one, we can see a high med value for the similarity. This is due to the fact that the matrix decomposition would descover deeper relations using the latent comparing with the relations extracted by the cosine similarity.

Now if we calcuate the agreement of the top 10 similar countries for each country between the two models.

In [133]:
```
intersection_length=10
modelnames=score_matrices.keys()
for m in range(len(modelnames)):
    modelname_x=modelnames[m]
    for n in range(len(modelnames)):
        if m>=n:
            continue
        modelname_y=modelnames[n]
        first_arg_sort=np.argsort(score_matrices[modelname_x],axis=1)
        second_arg_sort=np.argsort(score_matrices[modelname_y],axis=1)
        N=score_matrices[modelname_x].shape[0]
        res=[]
        for i in range(N):
            r=np.intersect1d(first_arg_sort[i,N-intersection_length:],second_ar
g_sort[i,N-intersection_length:])
            res.append(len(r))
        print '{}-{} Agreement percentage :{}%'.format(modelname_x,modelname_y,
np.round((np.mean(res)/intersection_length)*100))
```

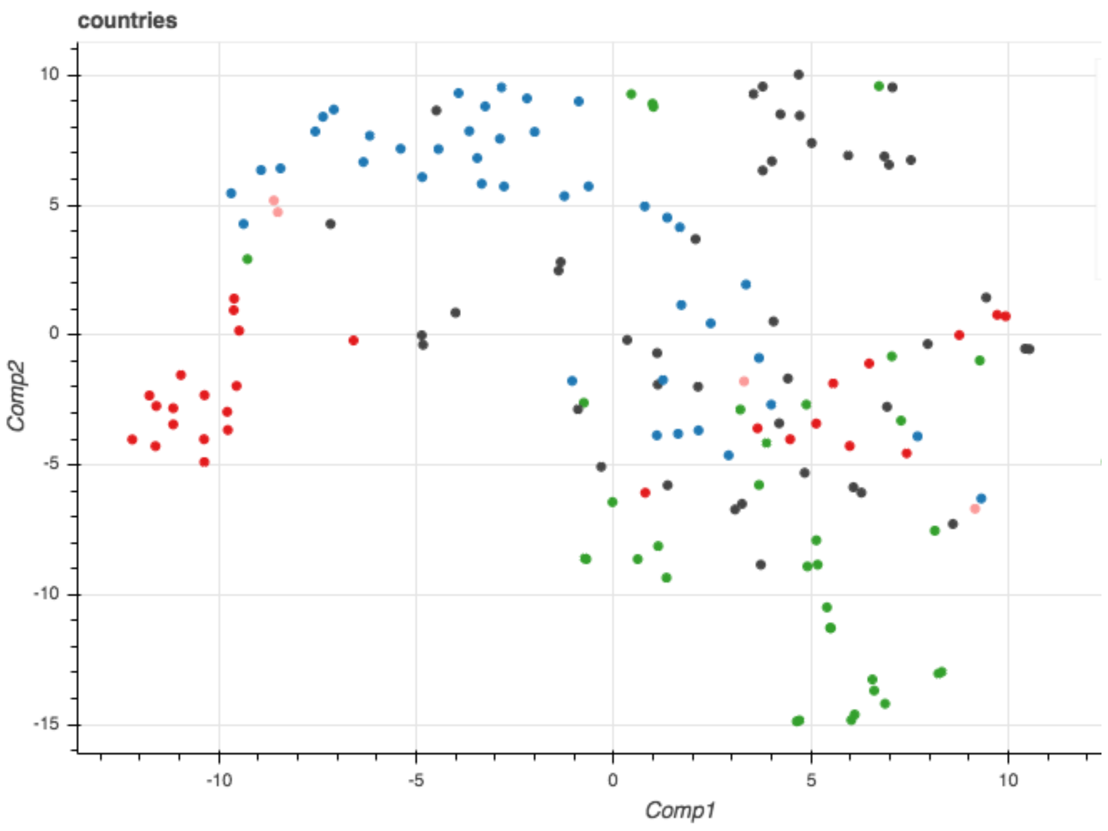
```
tfidf-cosine Agreement percentage :52.0%
tfidf-bpr Agreement percentage :41.0%
tfidf-als Agreement percentage :58.0%
cosine-bpr Agreement percentage :32.0%
cosine-als Agreement percentage :48.0%
bpr-als Agreement percentage :39.0%
```

The considerable degree of agreement between the tfidf and cosine results is expected as the both models are distance based and have similar characterstics. However, the high degree of agreement between tfidf and als is what's intereting and need more investigation in order to see what kind of semnetic aspects each model capture. bpr has different results with less than 40% agreement with the other models.

To check if there is a similarity or difference pattern in the data, we visualize the data into two dimensional space. This is done by taking the similarity score matrix and using t-sne visualization, we project the countries that reserve the neighbours relations. The colors of the scattered points represent the contenient of the country.

first we start with the tfidf model

```
In [135]: show_country_similarity(score_matrices['tfidf'],countries)
```



As we can notice in the figure there is fairly distinguish clustering of the data accoding to the contenient.

Intersting observations:

1- The geographical distance between the countries is fairly refected in the distance in the visualisation (ex. Nordic counties, noth and south american, East and west Asian countries, North African).

2- Australia and NewZealand are well fit into the Europian cluster.

3- South Africa is placed between the Europian cluster and the America cluster.

4- North of Africa countries fit close to the Europian cluster.

5- An isolated small group of 6 mix countries are also isolated with one event reported (DEATH).

6- In middle mix area where it is hard to find a predicted pattern in the cluster.

We can a simplistic and primitive explaination for the observation as:

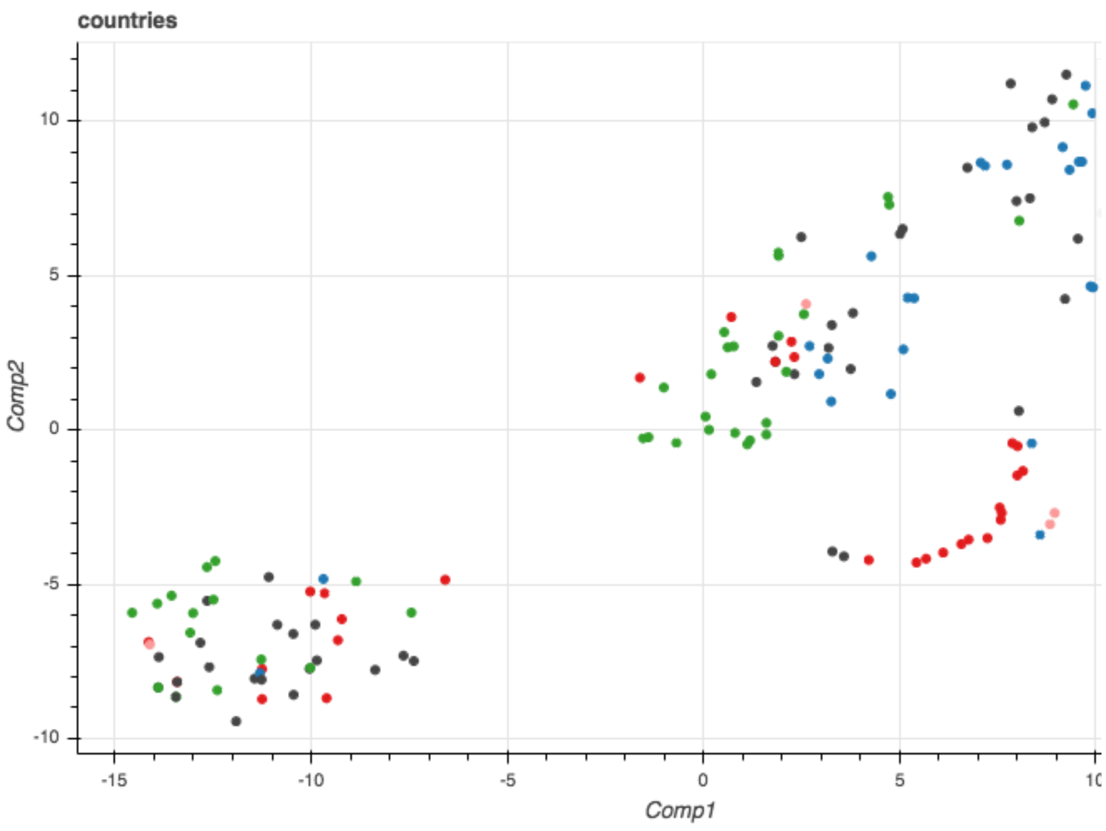
1- The genetic information plays a considerable role in deciding the events observed.

2- The geographical position plays a role in which deseases are more likely to appeares in certain area, and as a result, certain drugs are more consumed which associated with certain adverse events.

3- The healthcare system in the certain areas in the world is more advance comparing with others and that could reflect as different reporting.

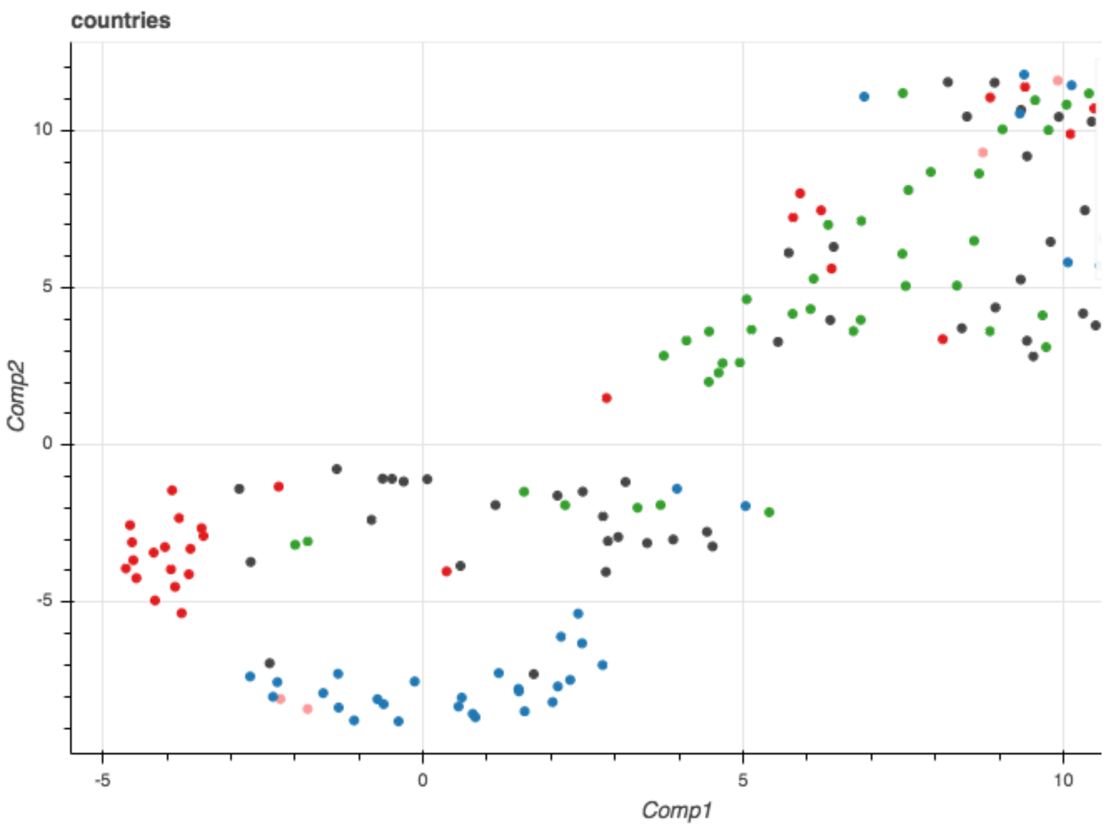
To check if any more observation we can extract from the other models, we will mention only the different observations from the other graphs if found.

```
In [136]: show_country_similarity(score_matrices['cosine'],countries)
```



In this one we can see a different distribution for the clusters where interestingly, UK, Ireland, Australia and New Zealand are close to USA and Canada. Also, we notice, some Middle east (Saudi arabia, Kuwit, Israel, and lebanon) countries are close to Greece, Romania and Hungary. A big mixed group with event DEATH as most frequent event.

```
In [137]: show_country_similarity(score_matrices['bpr'],countries)
```



In addition to the previously mention, it is worth to mention:

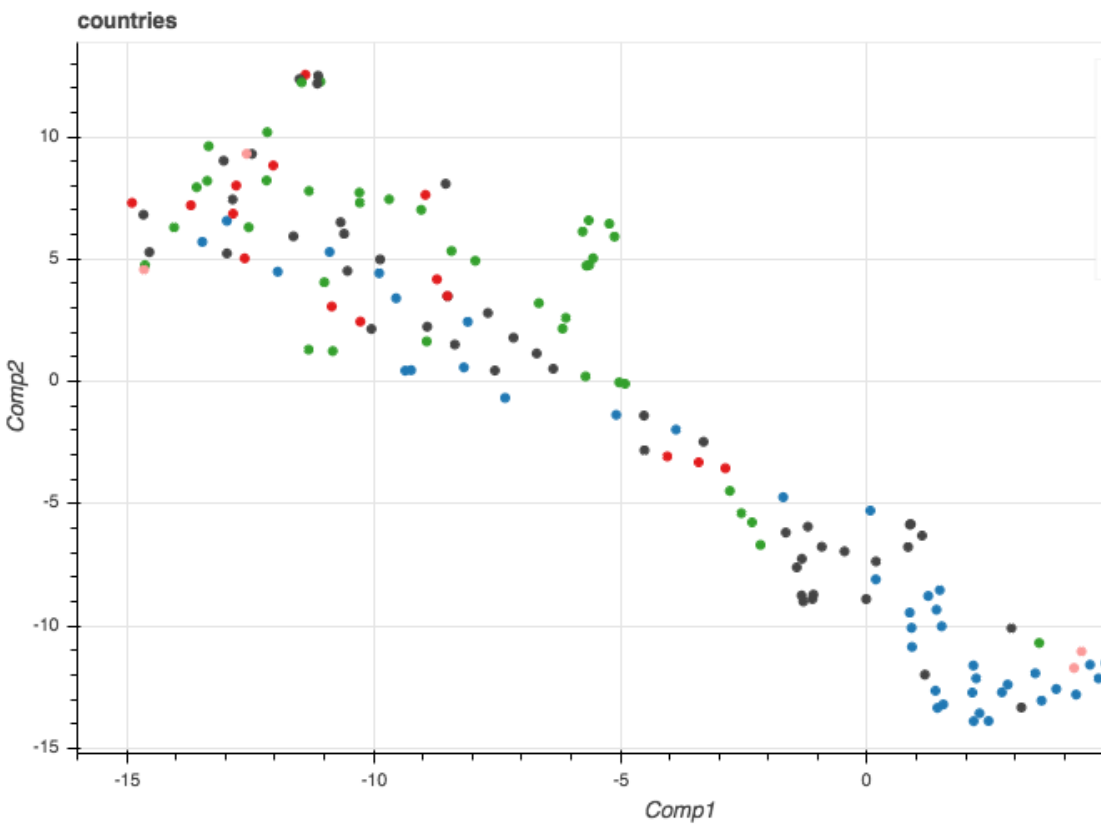
Lebanon appears in the south american cluster (which could be explained as the high level of migrants from Lebanon are based in South America)

East Europe Form a new and seperated cluster.

A larger mixed group with very few reported events (1 or 2)

Several American countries appears in Africa Like Nicaragua, Jamaica,Frensh Guiana

In [138]:
```
show_country_similarity(score_matrices['als'],countries)
```



In order to find out if such pattern are repeatable on drug level, we repeat the visualization for a certain drug. we choose random one of the top 20 drug those have the highest reporting cases.

In [139]:

```
# In order to find out if such pattern are repeatable on drug level, we repeat
the visualization for a certain drug.
# we choose random one of the top 20 drug those have the highest reporting case
s.
# The input file
file_name='./data/PREGABALIN_events.csv'
# number of factors for the MD approaches
num_of_factors=300
# use the raw data or a ratio per country
ratio=True
# use binary or float (True means there is event reported in that country witho
ut how many)
normalized=False
# The minimum number of reports to consider per event
threshold=1
#  the maximum number of events to consider per country
number_of_events_per_country=300
# create the dataset
sparse_mat,countries,events,df=file_to_sparse_matrix(file_name,number_of_events
_per_country,threshold,normalized,ratio)
# the CF approaches to compare
models_to_examine=["als","bpr","cosine","tfidf"]
models={}
score_matrices={}
for modelname in models_to_examine:
    model=create_fit_CF_model(modelname,sparse_mat,num_of_factors)
    models[modelname]=model
    score_matrix=get_country_matrix_similarity(models[modelname],countries)
    score_matrices[modelname]=np.asarray(score_matrix)
```

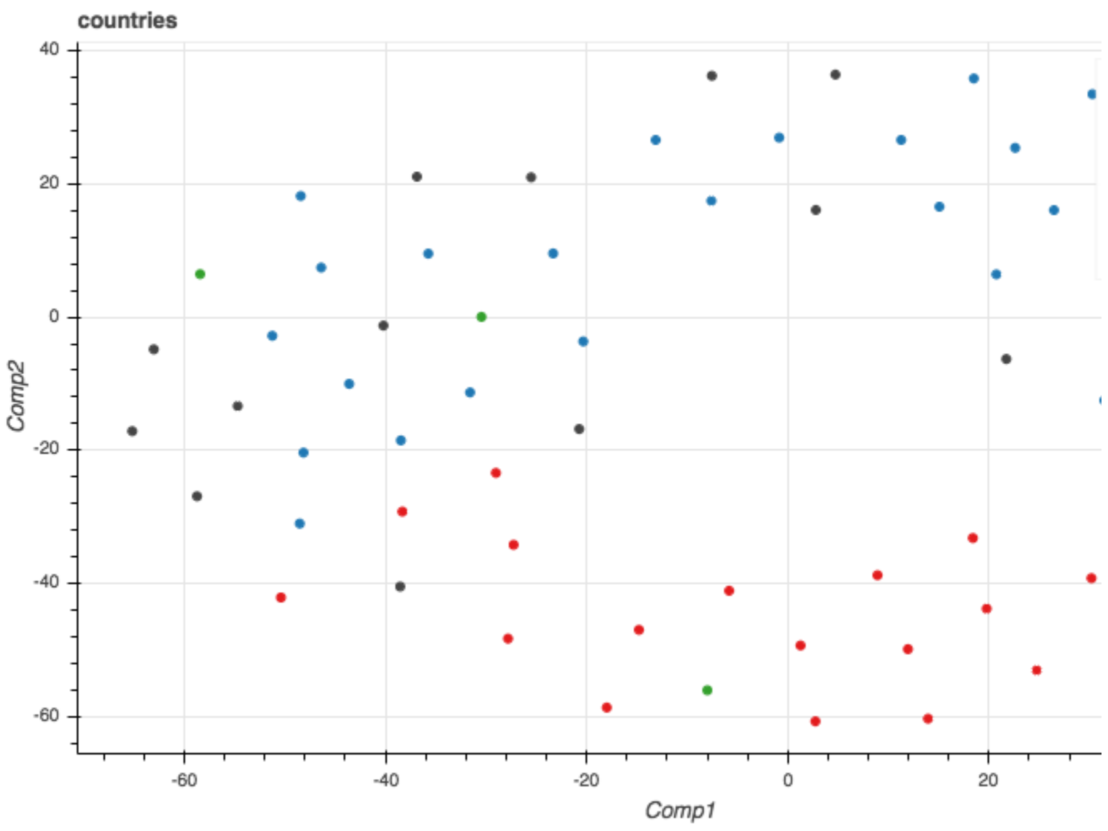
```
/Users/mohamedabou-zleikha/venv/lib/python2.7/site-packages/ipykernel_launcher
.py:14: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/sta
ble/indexing.html#indexing-view-versus-copy

100%|██████████| 15.0/15 [00:01<00:00, 14.90it/s]
100%|██████████| 100/100 [00:01<00:00, 93.23it/s, skipped=44.83%, correct=78.4
6%]
100%|██████████| 62/62 [00:00<00:00, 10011.43it/s]
100%|██████████| 62/62 [00:00<00:00, 25524.82it/s]
```

In [140]:

```
for modelname in score_matrices:
    score_matrix1=np.asarray(score_matrices[modelname])
    score_matrix1=(score_matrix1-score_matrix1.min())/(score_matrix1.max()-scor
e_matrix1.min())
    medV,meanV,maxV,minV,stdV,mean_num_outliers=get_outliers(score_matrix1,coun
tries)
    print modelname
    print 'Med : {}, Std : {}, Max : {}, Min: {},Avg Outliers: {}'.format(medV.
score.mean(),stdV.score.mean(),maxV.score.mean(),minV.score.mean(),mean_num_out
liers[0])
```

```
tfidf
Med : 0.0, Std : 0.0739040157538, Max : 0.260728018951, Min: 0.0,Avg Outliers:
3
cosine
Med : 0.0, Std : 0.18335800893, Max : 0.547085919543, Min: 0.0,Avg Outliers: 4
bpr
Med : 0.598521889458, Std : 0.264629737072, Max : 0.948636177086, Min: 0.0,Avg
Outliers: 0
als
Med : 0.177100478641, Std : 0.0798418910606, Max : 0.372453316806, Min: 0.0,Av
g Outliers: 0
```

```
In [143]: show_country_similarity(score_matrices['bpr'],countries)
```



However, For other drugs, no visible pattern could recognized but with different similar events per country as the case:

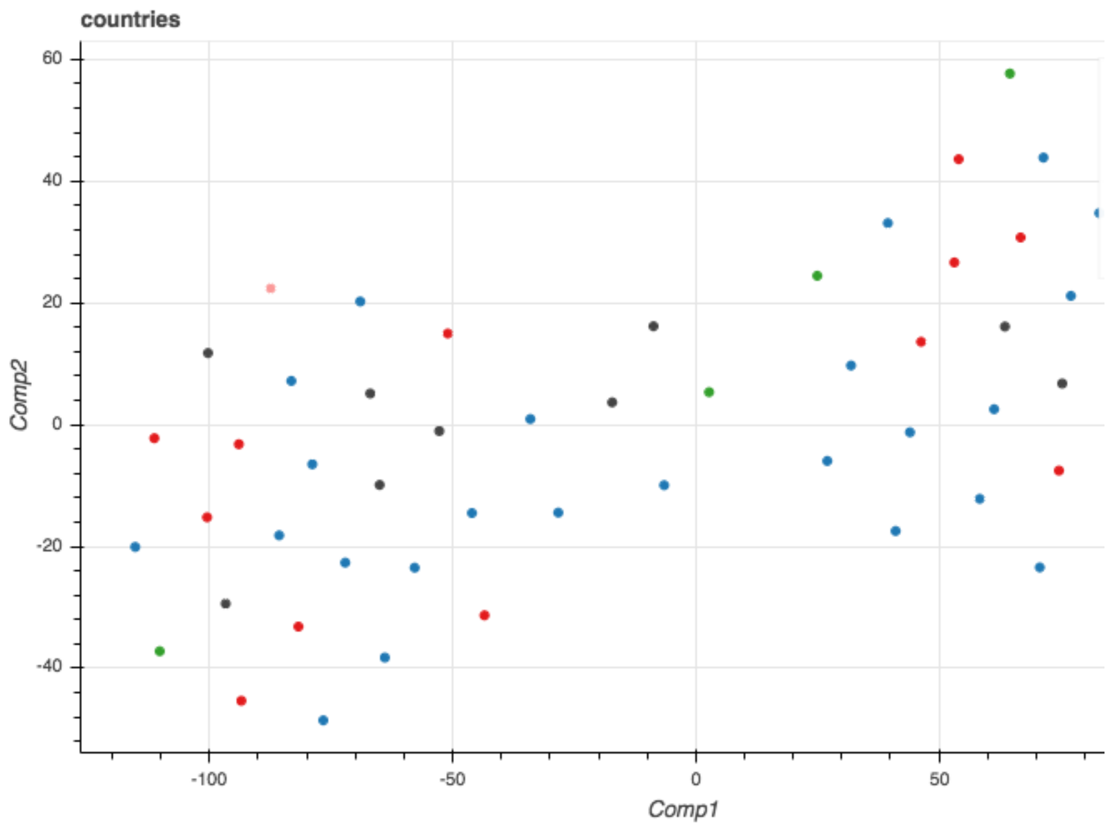
```
In [144]: #
file_name='./data/INTERFERON BETA-1A_events.csv'
# number of factors for the MD approaches
num_of_factors=100
# use the raw data or a ratio per country
ratio=False
# use binary or float (True means there is event reported in that country witho
ut how many)
normalized=False
# The minimum number of reports to consider per event
threshold=1
#  the maximum number of events to consider per country
number_of_events_per_country=50
# create the dataset
sparse_mat,countries,events,df=file_to_sparse_matrix(file_name,number_of_events
_per_country,threshold,normalized,ratio)
# the CF approaches to compare
models_to_examine=["als","bpr","cosine","tfidf"]
models={}
score_matrices={}
for modelname in models_to_examine:
    model=create_fit_CF_model(modelname,sparse_mat,num_of_factors)
    models[modelname]=model
    score_matrix=get_country_matrix_similarity(models[modelname],countries)
    score_matrices[modelname]=np.asarray(score_matrix)
```

```
100%|██████████| 15.0/15 [00:00<00:00, 218.19it/s]
100%|██████████| 100/100 [00:00<00:00, 155.90it/s, skipped=34.72%, correct=59.
89%]
100%|██████████| 59/59 [00:00<00:00, 23993.01it/s]
100%|██████████| 59/59 [00:00<00:00, 31463.95it/s]
```

In [145]:
```
for modelname in score_matrices:
    score_matrix1=np.asarray(score_matrices[modelname])
    score_matrix1=(score_matrix1-score_matrix1.min())/(score_matrix1.max()-scor
e_matrix1.min())
    medV,meanV,maxV,minV,stdV,mean_num_outliers=get_outliers(score_matrix1,coun
tries)
    print modelname
    print 'Med : {}, Std : {}, Max : {}, Min: {},Avg Outliers: {}'.format(medV.
score.mean(),stdV.score.mean(),maxV.score.mean(),minV.score.mean(),mean_num_out
liers[0])
```

```
tfidf
Med : 0.0, Std : 0.0637358676032, Max : 0.240554780259, Min: 0.0,Avg Outliers:
0
cosine
Med : 0.0, Std : 0.168237373858, Max : 0.490760513452, Min: 0.0,Avg Outliers:
0
bpr
Med : 0.470204099508, Std : 0.273693625185, Max : 0.873866152965, Min: 0.0,Avg
Outliers: 0
als
Med : 0.103841772524, Std : 0.0371735405265, Max : 0.219125759677, Min: 0.0,Av
g Outliers: 0
```

In [147]:
```
show_country_similarity(score_matrices['bpr'],countries)
```



In this task, we investigated the possibility of having different events for different countries. We found that there is significant difference between the events reported by different countries. We also found that there is similarity in the events reported from certain countries. This similarity could be connected to genetic or economical (healthcare quality) reasons. A more investigation is required where we can connect the data with the genetic information and the economical information in order to find out the exact reason. One of the problems is the few reported events and countries with few reported events. A more investigation is required to see the effect of removing such countries/events on the similarity.