# eTransafe Concordance analysis

This is the use scenario that has been described by Thomas Steger-Hartmann in a publication with Matthew Clark. The idea is to compare animal observations with clinical observations for the various drugs

1. determine the drugs that have been used in the preclinical and the clinical domain
2. compare the individual PTs for preclinical and clinical
3. compute the concordance matrix
   1. aggregate the PT terms per PT, HLT, or SOC level
   2. for each drug, count how many terms are in the TP, FP, TN, FN bins
4. Visualize the matrix

(C) 2022 Erasmus University Medical Center, Rotterdam, The Netherlands Author: Erik M. van Mulligen, [e.vanmulligen@erasmusmc.nl](mailto:e.vanmulligen@erasmusmc.nl)

import os  
import sys  
module\_path = os.path.abspath(os.path.join('..'))  
if module\_path not in sys.path:  
 sys.path.append(module\_path)  
from knowledgehub.api import KnowledgeHubAPI  
from Concordance.condordance\_utils import getClinicalDatabases, getPreclinicalDatabases  
from Concordance.mapper import Mapper  
  
import ipywidgets as w  
from IPython.display import display, Javascript  
from ipypublish import nb\_setup  
import numpy as np  
import mysql.connector  
  
import warnings  
warnings.filterwarnings('ignore')

api = KnowledgeHubAPI(server='DEV', client\_secret='3db5a6d7-4694-48a4-8a2e-e9c30d78f9ab')  
mapper = Mapper(api)

## Authenticate for KnowledgeHub

username = w.Text(value='tester',placeholder='Knowledge Hub account', description='username:', disabled=False)  
password = w.Password(value='', placeholder='Knowledge Hub password', description='password:', disabled=False)  
loginBtn = w.Button(description='Login')  
status = w.Output()  
  
def on\_button\_clicked(\_):  
 if not api.login(username.value, password.value):  
 print("Failed to login")  
 else:  
 print("successfully logged in")  
  
  
loginBtn.on\_click(on\_button\_clicked)  
w.VBox([username, password, loginBtn])

## Authenticate for the data stored in the database

global db  
  
dbhost = w.Text(value='localhost',placeholder='database host', description='host:', disabled=False)  
dbdatabase = w.Text(value='concordance-20220524',placeholder='database name', description='database:', disabled=False)  
dbusername = w.Text(value='root',placeholder='database username', description='username:', disabled=False)  
dbpassword = w.Password(value='', placeholder='database password', description='password:', disabled=False)  
dbLoginBtn = w.Button(description='Login')  
status = w.Output()  
  
def dbLoginBtn\_click(\_):  
 global db  
 try:  
 db = mysql.connector.connect(host=dbhost.value, database=dbdatabase.value, user=dbusername.value, password=dbpassword.value)  
 print("successfully logged in database")  
 except Exception as e:  
 print("failed to log in database")  
 sys.exit(0)  
   
dbLoginBtn.on\_click(dbLoginBtn\_click)  
w.VBox([dbhost, dbdatabase, dbusername, dbpassword, dbLoginBtn])

## The database

In order to be able to compute the concordance tables we have constructed a database with all preclinical and clinical findings found for drugs (i.e. inchikeys) that are both found in the preclinical and clinical data. For the preclinical data we restricted to findings that are treatment related and not in the control group. For each of the preclinical findings we checked with the semantic service whether the equivalent clinical finding was found in the clinical findings and vice versa. This is stored in the database as 'mapped' per finding. For each of the preclinical and clinical findings we derived the MedDRA PT it is related to. For preclinical terms this is done through the semantic service. For clinical terms the findings are already expressed as MedDRA PT terms. These mappings are stored in the database as 'PT' per finding.

## Drug mapping

We maintain a list of drugs that can be found in the preclinical and clinical data with its inchi-group (inchi-group being the first 14 characters of an inchi-key). Per drug we have stored the finding ids that are associated per database.

cursor = db.cursor()  
cursor.execute('SELECT inchi\_group, inchi\_keys, names FROM drugs')  
drugs = [{'inchi\_group': r[0], 'inchi\_keys': [i for i in r[1].split(',')], 'names':[i for i in r[2].split(',')]} for r in cursor.fetchall()]  
print(f'{len(drugs)} drugs found')

## Overview of the drugs

pd = nb\_setup.setup\_pandas(escape\_latex=False)  
df = pd.DataFrame(np.random.rand(len(drugs),3),columns=['inchi\_group','inchi\_key','name'])  
df.inchi\_group = [drug['inchi\_group'] for drug in drugs]  
df.inchi\_key = [drug['inchi\_keys'][0] for drug in drugs]  
df.name = [drug['names'][0] for drug in drugs]  
df.round(3)

## Concordance table

Per drug retrieve the preclinical and clinical PT terms.

* true positives are the PT terms that can be found present in the preclinical and clinical data per drug.
* false positives are the PT terms that can be found in the preclinical data but not in the clinical data per drug
* false negatives are the PT terms that can not be found in the preclinical data but in the clinical data per drug
* true negatives are all PT terms that can not be found in the preclinical data and in the clinical data per drug

from Concordance.meddra import MedDRA  
  
level = 'soc'  
pt\_to\_group = {}  
  
def getPTDrugFindings(db, drug, clinical):  
 cursor = db.cursor()  
 cursor.execute(f'SELECT DISTINCT finding\_code FROM findings WHERE inchi\_group = "{drug}" AND clinical = {clinical} AND distance IS NOT NULL')  
 return [r[0] for r in cursor.fetchall()]  
  
def getAllPreclinicalClinicalDistances(db):  
 cursor = db.cursor()  
 cursor.execute('SELECT finding\_code, min(distance) FROM findings WHERE distance is not NULL GROUP BY finding\_code')  
 return {finding[0]: finding[1] for finding in cursor.fetchall()}  
  
def getAllPreClinicalClinicalPTs(db):  
 cursor = db.cursor()  
 cursor.execute(f'SELECT distinct finding\_code FROM findings WHERE distance is not NULL')  
 return [finding[0] for finding in cursor.fetchall()]  
  
def getGroup(meddra, pt, level):  
 if not pt in pt\_to\_group:  
 if level == 'pt':  
 group = meddra.getPt(pt)  
 elif level == 'hlt':  
 group = meddra.getHLT(pt)  
 elif level == 'soc':  
 group = meddra.getSoc(pt)  
 pt\_to\_group[pt] = list(group.keys())[0] if len(group) > 0 else None  
 return pt\_to\_group[pt]  
  
print(f'username={dbusername.value}, password={dbpassword.value}')  
meddra = MedDRA(username=dbusername.value, password=dbpassword.value)  
ClinicalDatabases = getClinicalDatabases(api);  
PreclinicalDatabases = getPreclinicalDatabases(api);  
  
groups = {}  
preclinical\_pts = {}  
clinical\_pts = {}  
for drug in drugs:  
 inchi\_group = drug['inchi\_group']  
 preclinical\_pts[inchi\_group] = set([getGroup(meddra, pt, level) for pt in getPTDrugFindings(db=db, drug=inchi\_group, clinical=0)])  
 clinical\_pts[inchi\_group] = set([getGroup(meddra, pt, level) for pt in getPTDrugFindings(db=db, drug=inchi\_group, clinical=1)])  
  
all\_preclinical\_clinical\_pts = set([getGroup(meddra, pt, level) for pt in getAllPreClinicalClinicalPTs(db=db)])  
all\_preclinical\_clinical\_distances = {getGroup(meddra, pt, level): distance for (pt, distance) in getAllPreclinicalClinicalDistances(db=db).items()}  
  
for code in all\_preclinical\_clinical\_pts:  
 #group = getGroup(meddra, code, level)  
 group = code  
  
 if group is not None:  
 if not group in groups:  
 groups[group] = {'tp': 0, 'fp': 0, 'fn': 0, 'tn': 0, 'drugs': [], 'distance': all\_preclinical\_clinical\_distances[code]}  
 elif abs(groups[group]['distance']) > abs(all\_preclinical\_clinical\_distances[code]):  
 groups[group]['distance'] = all\_preclinical\_clinical\_distances[code]  
  
 for drug in drugs:  
 inchi\_group = drug['inchi\_group']  
 if inchi\_group not in groups[group]['drugs']:  
 groups[group]['drugs'].append(inchi\_group)  
 if code in preclinical\_pts[inchi\_group]:  
 if code in clinical\_pts[inchi\_group]:  
 groups[group]['tp'] += 1  
 else:  
 groups[group]['fp'] += 1  
 else:  
 if code in clinical\_pts[inchi\_group]:  
 groups[group]['fn'] += 1  
 else:  
 groups[group]['tn'] += 1

## Concordance tables

from Concordance.condordance\_utils import getName  
  
def compute\_lrp(group):  
 sensitivity = compute\_sensitivity(group)  
 specificity = compute\_specificity(group)  
 if specificity is not None and sensitivity is not None:  
 return sensitivity / (1 - specificity) if specificity != 1 else None  
 else:  
 return None  
  
def compute\_lrn(group):  
 sensitivity = compute\_sensitivity(group)  
 specificity = compute\_specificity(group)  
 if specificity is not None and sensitivity is not None:  
 return (1 - sensitivity) / specificity if specificity != 0 else None  
 else:  
 return None  
   
def compute\_chisquare(group):  
 tp = group['tp']  
 fp = group['fp']  
 fn = group['fn']  
 tn = group['tn']  
 total = tp + fp + fn + tn  
 e11 = ((tp + fp) \* (tp + fn)) / total  
 e12 = ((tp + fp) \* (fp + tn)) / total  
 e21 = ((fn + tn) \* (tp + fn)) / total  
 e22 = ((fn + tn) \* (fp + tn)) / total  
 try:  
 return (((tp - e11)\*\*2)/e11) + (((fp - e12)\*\*2)/e12) + (((fn - e21)\*\*2)/e21) + (((tn - e22)\*\*2)/e22)  
 except Exception as e:  
 return None  
  
def compute\_sensitivity(group):  
 tp = group['tp']  
 fn = group['fn']  
 return tp / (tp + fn) if (tp + fn) > 0 else None  
  
def compute\_specificity(group):  
 fp = group['fp']  
 tn = group['tn']  
 return tn / (fp + tn) if (fp + tn) > 0 else None  
  
group\_title = 'MedDRA ' + level.upper()  
pd.set\_option('display.max\_rows', None)  
pd.set\_option('display.colheader\_justify', 'left')  
pd.options.display.float\_format = '{:.2f}'.format  
df = pd.DataFrame(np.random.rand(len(groups),11),columns=[group\_title,'min.distance', 'TP','FP', 'FN', 'TN', 'Sensitivity', 'Specificity', 'LR+', 'LR-', 'chi-square'])  
df[group\_title] = [getName(meddra, code, level) for code in groups]  
df['min.distance'] = [groups[code]['distance'] for code in groups]  
df.TP = [groups[code]['tp'] for code in groups]  
df.FP = [groups[code]['fp'] for code in groups]  
df.FN = [groups[code]['fn'] for code in groups]  
df.TN = [groups[code]['tn'] for code in groups]  
df['Sensitivity'] = [compute\_sensitivity(groups[code]) for code in groups]  
df['Specificity'] = [compute\_specificity(groups[code]) for code in groups]  
df['LR+'] = [compute\_lrp(groups[code]) for code in groups]  
df['LR-'] = [compute\_lrn(groups[code]) for code in groups]  
df['chi-square'] = [compute\_chisquare(groups[code]) for code in groups]  
df.round(3)  
df = df.sort\_values(by=['LR+'], ascending=False)  
dfStyler = df.style.set\_properties(\*\*{'text-align': 'right'})  
dfStyler.set\_table\_styles([dict(selector='th', props=[('text-align', 'left')])])  
dfStyler.set\_properties(subset=[group\_title], \*\*{'text-align': 'left'}).hide\_index()

html = df.to\_html(index=False, justify='right', border=1)  
  
#write html to file  
text\_file = open("../data/concordance\_" + level.upper() + ".html", "w")  
text\_file.write(html)  
text\_file.close()