ACDA Documentation

Release 0.0.1

S. Domanskyi **A. Srivastava**

Apr 25, 2023

CONTENTS

ACDA (Augmented Cancer Drug Atlas) is a software tool for prioritizing drug pairs that have synergystic effect when used in combination.

Note: ACDA is introduced in:

https://biorxiv.org

CHAPTER

ONE

CONTENTS

1.1 Overview

This documentation describes Augmented Cancer Drug Atlas (ACDA) drug synergy prediction methods. We augmented the drug synergy prediction modeling approach CDA described in Narayan et al. by applying a Random Forest Regression and optimization via cross-validation hyperparameter tuning. For ease of sharing and use we implemented it as a python package. The source code is located at https://github.com/TheJacksonLaboratory/drug-synergy.

1.1.1 Description of the package functionality

This documentation shows what data inputs are necessary and how to use the ACDA API (Application Programming Interface) to generate the synergy predictions and the corresponding visualizations.

1.1.2 Versions change log

0.0.1 Beta release

0.0.2 Added examples and adjusted the package codebase

0.0.3 Update of the train-test split function

0.0.4 Minor code edits to allow to lowercase drug names in preparation functions

0.0.5 Added stratified splitting for cross-validation, added method that averages predictions of ACDA and EN-ACDA

0.0.6 Added a function that averages prediction for Drug A/Drug B and Drug B/Drug A

1.2 Getting Started

These instructions will get you a copy of the project up and running on your machine for data analysis, development or testing purposes.

1.2.1 Installation

Install of the latest release of `acda`:

```
$ pip install acda
```

For detailed instructions and other ways to install `acda` as well as list of optional packages and instructions on how to install them see **Prerequisites** section at https://github.com/TheJacksonLaboratory/acda

1.2.2 Loading the package

Start an interactive Python session or, alternatively, in your script, import the package:

```
import acda
```

1.3 Description of the API

1.3.1 Description of the main package functionality

Functions:

MonteCarloCrossValidation(dfTas[, n, ...])	For continuous predictee: clf_for_CDA=LinearRegression()
averagePredictionPairs(se)	
downsampleRun(ra[, dfTas, pref, mid, rep, ...])	df_sub10000 = downsampleRun(range(500, 10000, 500), dfTas=dfTas_AZ_breast, pref='', mid='AZ_breast')
filter_synergy_pairs_list(df, dfTa[, pref])	Get pairs that overlap with sensitivity-mutations-targets data
fit_validate_predict(inData, inSynergy[, ...])	
getCDAdrugDistance(df_drugs_celllines[, method])	Excludes pairs of points with missing data method: {'pearson', 'kendall', 'spearman', 'cosine'}
getDistanceAndSensitivityData(df[, method, ...])	Sensitivity data and CDA drug distance dfC, dfS, Z = getDistanceAndSensitivity-Data(df_drug_sensitivity_GDSC1)
makeCDAformattedData(tissue, ...[, ...])	
prepDfTa(df_drug_sensitivity, ...[, lower])	Drug targeting mutated genes
prepDfTas(dfTa, dfKS, se_tissue_annotation)	Reshape dfTa, add tissue and synergy information
prepareFromAZforCDA(dir, ...[, fname, ...])	{'IC50', 'LNIC50', 'H'}
prepareFromDCforCDA(df_DC, study, tissue, ...)	
prepareFromDCfull(df_temp[, sample, ...])	
sample_train_predicted(df_G2, random_state)	
split_train_test_validate_predict(df[, ...])	
testCase(*args, **kwargs)	
trainOneTestAnother(df_one, df_another[, n, ...])	
tryExcept(func)	

getCDAdrugDistance(*df_drugs_celllines*, *method='pearson'*)
Excludes pairs of points with missing data method: {'pearson', 'kendall', 'spearman', 'cosine'}

getDistanceAndSensitivityData(*df*, *method='pearson'*, *sensitivity_metric='LNIC50'*, *drug='DRUG'*, *lower=True*)
Sensitivity data and CDA drug distance dfC, dfS, Z = getDistanceAndSensitivity-Data(df_drug_sensitivity_GDSC1)

prepDfTa(*df_drug_sensitivity*, *se_models_mutations*, *se_drug_targets*, *dname*, *lower=True*)
Drug targeting mutated genes

filter_synergy_pairs_list(*df*, *dfTa*, *pref='Synergy pairs'*)
Get pairs that overlap with sensitivity-mutations-targets data

Usage: dfKS = filter_synergy_pairs_list(df_drug_synergy_Narayan, dfTa)

prepDfTas(*dfTa*, *dfKS*, *se_tissue_annotation*, *tissue=None*)
Reshape dfTa, add tissue and synergy information

split_train_test_validate_predict(*df*, *factor=0.5*, *random_state=None*, *stratified=False*)

makeCDAformattedData(*tissue*, *se_drug_synergy*, *se_tissue_annotation*, *df_drug_sensitivity*, *se_models_mutations*, *se_drug_targets*, *dataset*, *sensitivityMethod='cosine'*, *sensitivity_metric='LNIC50'*, *returnMore=False*, *lower=True*)

prepareFromDCforCDA(*df_DC*, *study*, *tissue*, *se_models_mutations*, *sample=None*, *random_state=None*, *sensitivity_measure='AUC'*, *synergy='SYNERGY_ZIP'*, *returnMore=False*)

prepareFromDCfull(*df_temp*, *sample=None*, *random_state=None*, *returnMore=False*)

prepareFromAZforCDA(*dir*, *se_tissue_annotation*, *se_models_mutations*, *se_drug_targets*, *fname='oi_combinations_synergy_scores_final.txt'*, *sensitivity_measure='LNIC50'*)
{'IC50', 'LNIC50', 'H'}

trainOneTestAnother(*df_one*, *df_another*, *n=10*, *n_sample=0.5*)

averagePredictionPairs(*se*)

fit_validate_predict(*inData*, *inSynergy*, *extData=None*, *extSynergy=None*, *cv=None*, *max_iter=10000*, *clf=None*, ***kwargs*)

sample_train_predicted(*df_G2*, *random_state*, *sample=100*)

tryExcept(*func*)

testCase(**args*, ***kwargs*)

MonteCarloCrossValidation(*dfTas*, *n=10*, *sample_non_synergy=False*, *sample_non_synergy_size=100*, *clf_for_CDA=LogisticRegression()*, *deidentify=False*, *factor=0.6666666666666666*, *stratified=False*)
For continuous predictee: clf_for_CDA=LinearRegression()

downsampleRun(*ra*, *dfTas=None*, *pref='temp'*, *mid='temp'*, *rep=10*, *basedir='output/'*, *cv=None*, *stratified=False*)
df_sub10000 = downsampleRun(range(500, 10000, 500), dfTas=dfTas_AZ_breast, pref='', mid='AZ_breast')

1.3.2 Description of the auxiliary package functionality

Functions:

centerDf(df)	similar to the StandardScaler: from sklearn.preprocessing import StandardScaler StandardScaler().fit(df).transform(df)
convertProteinNamesToGenes(se, dfgp)	
encodeNames(df)	A.k.a.
fetchMySQL(s[, host, database, user, password])	
getGeneToProteinNameAssociation([host, ...])	Tables in uniProt: ['accToKeyword', 'accToTaxon', 'author', 'bigFiles', 'citation', 'citationRc', 'citationRp', 'comment', 'commentType', 'commentVal', 'commonName', 'description', 'displayId', 'extDb', 'extDbRef', 'feature', 'featureClass', 'featureId', 'featureType', 'gene', 'geneLogic', 'history', 'info', 'keyword', 'organelle', 'otherAcc', 'pathogenHost', 'protein', 'proteinEvidence', 'proteinEvidenceType', 'rcType', 'rcVal', 'reference', 'referenceAuthors', 'tableDescriptions', 'tableList', 'taxon', 'varAcc', 'varProtein']
ro_normalized(df)	Takes a dataframe where index has 3 levels (MODEL, DRUG1, DRUG2), and there are two columns, first with ground truth scores, second with predicted scores.

encodeNames (*df*)
A.k.a. One-hot encoding ['MODEL', 'DRUG1', 'DRUG2'] should be present either in index levels or in the columns. The idea is from the AstraZeneca DREAM challenge second-best winning method for drug synergy prediction.

ro_normalized (*df*)
Takes a dataframe where index has 3 levels (MODEL, DRUG1, DRUG2), and there are two columns, first with ground truth scores, second with predicted scores.

Output is the average pearson correlation coefficient for each drug pair weighted by the number of models for which a pair was measured against.

See definitions in: https://www.synapse.org/#!Synapse:syn4231880/wiki/235660

fetchMySQL (*s*, *host='host'*, *database='database'*, *user='user'*, *password='password'*)

getGeneToProteinNameAssociation (*host='genome-mysql.cse.ucsc.edu'*, *database='uniProt'*, *user='genomep'*, *password='password'*)
Tables in uniProt: ['accToKeyword', 'accToTaxon', 'author', 'bigFiles', 'citation', 'citationRc', 'citationRp', 'comment', 'commentType', 'commentVal', 'commonName', 'description', 'displayId', 'extDb', 'extDbRef', 'feature', 'featureClass', 'featureId', 'featureType', 'gene', 'geneLogic', 'history', 'info', 'keyword', 'organelle', 'otherAcc', 'pathogenHost', 'protein', 'proteinEvidence', 'proteinEvidenceType', 'rcType', 'rcVal', 'reference', 'referenceAuthors', 'tableDescriptions', 'tableList', 'taxon', 'varAcc', 'varProtein']

To see all columns in a table: fetchMySQL("SHOW COLUMNS FROM my_table;")

Usage: assoc = getGeneToProteinNameAssociation()

convertProteinNamesToGenes (*se*, *dfgp*)

centerDf (*df*)
similar to the StandardScaler: from sklearn.preprocessing import StandardScaler StandardScaler().fit(df).transform(df)

1.4 Input data format

Input is expected as spreadsheets of comma-separated values `csv`.

1.5 Full format

Full format can be specified as shown below. If a field is not known, e.g. "MODEL_MUTATIONS" for a given "MODEL", then leave it blank.

MODEL	DRUG1	DRUG2	TISSUE	MODEL_MUTATIONS	SENSITIVITY_DRUG1	SENSITIVITY_DRUG2	SYNERGY_SCORE	DRUG1_TARGETS	DRUG2_TARGETS
Hx147	CHEMBL3707345	AZD8931	lung	KRAS	40.498	41.787	0.898170	DAPK2, DAPK3, DYRK1B, DYRK4, DYRK6, FIST3, HI…	
NCI-H358	AZD1208	333419-42-0	lung	KRAS	3.643	25.140	4.306900		
NCI-H322	topotecan	AZ13453202	lung	TP53	24.558	20.683	2.752007		
NCI-H1975	841290-80-0	AZD6738	lung	EGFR, PIK3CA, TP53	18.982	5.044	-2.048408	ADCK4, AGMX1, ALK, ANKK1, ANKRD3, ARK5, ATK, …	FRAP, FRAP1, FRAP2, MTOR, RAFT1, RAPT1
NCI-H358	957054-30-7	NSC169534	lung	KRAS, TP53	13.152	24.732	3.957847	CLK2, CLK4, DYRK6, FIST3, FRAP, FRAP1, FRAP2,…	
…	…	…	…	…	…	…	…	…	…

1.6 Alternative data format (simple)

Alternatively, separate data tables can be specified. For example, GDSC is natively better presented in this format.

> *NB:* In the underlying software, MODEL, DRUG and GENE identifiers are used to match and query the tables described above. Therefore these identifiers must be curated so that they are consistent across the tables.

1.6.1 Known synergy pairs

The first three columns contain model identifier, and drugs identifiers. The last column is the synergy score, or, if unavailable, a binary value 1/0 to indicate synergy or no synergy between the pair of drugs.

MODEL	DRUG1	DRUG2	SYNERGY_SCORE
5637	cisplatin	sunitinib	1
5637	sunitinib	cisplatin	1
8505C	bortezomib	docetaxel	1
8505C	docetaxel	bortezomib	1
A172	cisplatin	temozolomide	1
...	...	...	...

1.6.2 Sensitivity

A table of sensitivity measures; examples include LNIC50, AUC etc.

MODEL	DRUG	LNIC50	AUC	...
201T	5-Fluorouracil	3.738474	0.873656	...
201T	A-83-01	5.577933	0.975815	...
201T	ACY-1215	2.008393	0.746972	...
201T	AGI-6780	2.031296	0.977958	...
201T	AICA Ribonucleotide	10.006271	0.972249	...
...	...	...	...	...

1.6.3 Model tissue annotation

We recommend to perform analysis by tissue. Therefore we require tissue annoation for each model.

MODEL	TISSUE
1181N1	Central Nervous System
1205Lu	Skin
1273-99	Soft Tissue
1321N1	Central Nervous System
143B	Bone
...	...

1.6.4 Model mutations

List of mutated genes for each model. Note that the truncated raw csv file looks as below:

```
` MODEL,MUTATED GENES 201T,"IL16, TEKT4P2, TP53, NDC1, MROH7, INTS11" 22RV1,
"SLC2A13, MUC19, SLC15A4, RAB40C, RHOT2" `
```

MODEL	GENES
201T	IL16, TEKT4P2, TP53, NDC1, MROH7, INTS11, ...
22RV1	SLC2A13, MUC19, SLC15A4, RAB40C, RHOT2, ...
23132-87	MYH7, EMILIN1, HECW2, CCDC39, PPARGC1B, ...
42-MG-BA	LRRC74A, PILRB, CUX1, RB1, STAG2, TP53, ...
451Lu	POTEG, SPEF1, LMBRD1, BRAF, ROBO2, TP53, ...
...	...

1.6.5 Drug targets

List of genes targeted by a drug for each drug.

DRUG	GENE
(5Z)-7-Oxozeaenol	MAP3K7
5-Fluorouracil	Antimetabolite (DNA & RNA)
A-443654	AKT1, AKT2, AKT3
A-770041	LCK, FYN
A-83-01	TGFB1
...	...

1.7 Examples

Example of loading and running a full-format dataset from DrugComb, AstraZeneca study, breast tissue. The script takes a few minutes per iteration (n=3 iterations are suggested for testing).

```
from acda.method_functions import *

sdatadir = '../docs/examples/data/'

df_full = pd.read_csv(
    sdatadir + 'DrugComb_ASTRAZENECA_breast.csv.gz').set_index(['MODEL', 'DRUG1',
↪'DRUG2', 'TISSUE'])

df, dfC, dfS, Z = prepareFromDCfull(df_full, 1000, returnMore=True, random_state=0)

df_DC_AZ_breast = MonteCarloCrossValidation(df, n=3)[0]
print(df_DC_AZ_breast)
```

	mean	sem
ACDA	0.867	0.028
CDA	0.521	0.039
EN	0.829	0.020
EN-ACDA	0.894	0.021

Example of Monte Carlo cross-validation on an alternative format dataset: GDSC2 subset of breast tissue with the CDA synergy pairs.

```
from acda.method_functions import *

sdatadir = '../docs/examples/data/'

df_drug_sensitivity_GDSC2 = pd.read_csv(
    sdatadir + 'GDSC2_drug_sensitivity.csv.gz').set_index(['MODEL', 'DRUG'])

se_drug_synergy_CDA = pd.read_csv(
    sdatadir + 'CDA_synergy_pairs.csv.gz').set_index(['MODEL', 'DRUG1', 'DRUG2'])[
↪'SYNERGY_SCORE']

se_tissue_annotation_GDSC2 = pd.read_csv(
    sdatadir + 'GDSC2_tissue_annotation.csv.gz').set_index(['MODEL'])['TISSUE']
```

(continues on next page)

(continued from previous page)

```
se_drug_targets_GDSC2 = pd.read_csv(
    sdatadir + 'GDSC2_drug_targets.csv.gz').set_index(['DRUG'])['TARGETS']

se_models_mutations_GDSC2 = pd.read_csv(
    sdatadir + 'GDSC2_model_mutations.csv.gz').set_index(['MODEL'])['MUTATIONS']

dfTas_GDSC2_breast = makeCDAformattedData('Breast',
                                          se_drug_synergy_CDA,
                                          se_tissue_annotation_GDSC2,
                                          df_drug_sensitivity_GDSC2,
                                          se_models_mutations_GDSC2,
                                          se_drug_targets_GDSC2,
                                          'GDSC2',
                                          sensitivity_metric='LNIC50')
print(dfTas_GDSC2_breast)

df_GDSC2_breast = MonteCarloCrossValidation(dfTas_GDSC2_breast, sample_non_
↪synergy=True)[0]
print(df_GDSC2_breast)
```

Example of generating predictions on the alternative format dataset and visualizing them on a heatmap.

```
from acda.method_functions import *
from acda.plot_functions import *

sdatadir = '../docs/examples/data/'

df_drug_sensitivity_GDSC2 = pd.read_csv(
    sdatadir + 'GDSC2_drug_sensitivity.csv.gz').set_index(['MODEL', 'DRUG'])

se_drug_synergy_CDA = pd.read_csv(
    sdatadir + 'CDA_synergy_pairs.csv.gz').set_index(['MODEL', 'DRUG1', 'DRUG2'])[
↪'SYNERGY_SCORE']

se_tissue_annotation_GDSC2 = pd.read_csv(
    sdatadir + 'GDSC2_tissue_annotation.csv.gz').set_index(['MODEL'])['TISSUE']

se_drug_targets_GDSC2 = pd.read_csv(
    sdatadir + 'GDSC2_drug_targets.csv.gz').set_index(['DRUG'])['TARGETS']

se_models_mutations_GDSC2 = pd.read_csv(
    sdatadir + 'GDSC2_model_mutations.csv.gz').set_index(['MODEL'])['MUTATIONS']

dfTas_GDSC2_breast, dfC_G2, dfS_G2, Z_G2 = makeCDAformattedData('Breast',
                                                                se_drug_synergy_CDA,
                                                                se_tissue_annotation_
↪GDSC2,
                                                                df_drug_sensitivity_
↪GDSC2,
                                                                se_models_mutations_
↪GDSC2,
                                                                se_drug_targets_GDSC2,
                                                                'GDSC2',
                                                                sensitivity_metric=
↪'LNIC50',
                                                                returnMore=True)
```

(continues on next page)

(continued from previous page)

```
se_predicted = pd.concat([sample_train_predicted(dfTas_GDSC2_breast, i) for i in␣
↪range(3)],
                         axis=1).unstack(0).groupby(level=1, axis=1).agg(np.nanmean).
↪stack(
                             ).reorder_levels([3, 0, 1, 2]).sort_index()
print(se_predicted)
se_predicted.to_csv('predicted.csv')

fig = plotHeatmapPredictedSynergy(dfC_G2, Z_G2, se_predicted[se_predicted>=0.95].
↪index.droplevel(-1).values)
fig.savefig('heatmap.png', dpi=300)

temp = dfTas_GDSC2_breast['SYNERGY_SCORE'].droplevel(['MODEL', 'TISSUE'])
fig = plotDendrogramWithKnownPairs(Z_G2, dfC_G2, temp[temp==1].index.unique())
fig.savefig('dendrogram.png', dpi=300)
```

Load data from DrugComb. The file "drugcomb_data_v1.5.csv" is 1.3Gb (compressed is only 170Mb) and can be downloaded from https://drugcomb.fimm.fi/jing/summary_v_1_5.csv.

```
from acda.general_functions import *

keepMonotherapyData = False

dir = '../data/'

if keepMonotherapyData:
    tempfname = dir + 'cacheDrugCombWithAllMono.pklz'
else:
    tempfname = dir + 'cacheDrugComb.pklz'

if not os.path.exists(dir):
    os.makedirs(dir)

if not os.path.isfile(tempfname):
    # Data was downloaded from "https://drugcomb.fimm.fi/jing/summary_v_1_5.csv"
    df_DC = pd.read_csv(dir + 'drugcomb_data_v1.5.csv.gz', index_col=0)
    df_DC = df_DC.set_index(['study_name', 'cell_line_name', 'drug_row', 'drug_col',
↪'tissue_name'])
    df_DC.index.names = ['STUDY', 'MODEL', 'DRUG1', 'DRUG2', 'TISSUE']

    # Remove entries which are duplicates
    df_DC = df_DC.sort_index(level='DRUG2', ascending=True)
    df_DC = df_DC.loc[~df_DC.index.duplicated(keep='first')]
    df_DC = df_DC.sort_index()

    # Remove entries where DRUG1 is equal to DRUG2
    df_DC = df_DC.loc[df_DC.index.to_frame()['DRUG1'] != df_DC.index.to_frame()['DRUG2
↪']]

    # Remove entries with no combinations measures, i.e. monotherapy experiments
    if not keepMonotherapyData:
        df_DC = df_DC.loc[pd.MultiIndex.from_frame(df_DC.index.to_frame().dropna())]

    dfgp = getGeneToProteinNameAssociation()
    convertProteinNamesToGenes(df_DC['drug_row_target_name'], dfgp)
```

(continues on next page)

(continued from previous page)

```
    convertProteinNamesToGenes(df_DC['drug_col_target_name'], dfgp)

    # Keep subset of the columns, rename selected, below is the list of "not selected
↪" columns:
    # ['conc_row_unit', 'conc_col_unit', 'css_row', 'css_col', 'css_ri', 'S_sum', 'S_
↪mean',
    # 'S_max', 'drug_row_clinical_phase', 'drug_col_clinical_phase']
    df_DC = df_DC[['ic50_row', 'ic50_col', 'ri_row', 'ri_col', 'synergy_zip',
↪'synergy_loewe',
                   'synergy_hsa', 'synergy_bliss', 'drug_row_target_name', 'drug_col_
↪target_name']]
    df_DC.columns = ['IC50_DRUG1', 'IC50_DRUG2', 'AUC_DRUG1', 'AUC_DRUG2', 'SYNERGY_
↪ZIP',
                     'SYNERGY_LOEWE', 'SYNERGY_HSA', 'SYNERGY_BLISS', 'DRUG1_TARGETS',
↪ 'DRUG2_TARGETS']

    df_DC.to_pickle(tempfname)
else:
    df_DC = pd.read_pickle(tempfname)

print(df_DC)
```

1.8 Description of the plotting function

Functions:

drawOneDownsampled(usedf, ax, panel, a, b, c, v)	
makeBarplotCrossDatasets(df_res_cross[, ...])	
makeBarplotSingleDatasets(df_res_single[, ...])	
plotDendrogramWithKnownPairs(Z, dfC, dfKS)	
plotHeatmapPredictedSynergy(dfC, Z, pvalues)	

plotDendrogramWithKnownPairs (*Z*, *dfC*, *dfKS*)

plotHeatmapPredictedSynergy (*dfC*, *Z*, *pvalues*)

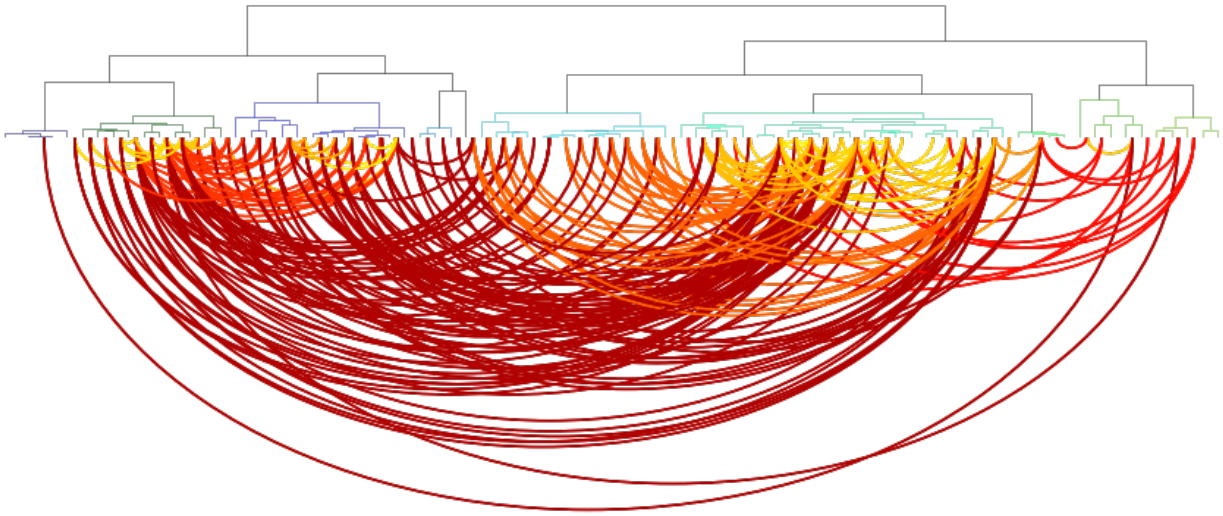
makeBarplotSingleDatasets (*df_res_single, figsize=(10, 8), c=['green', 'gold', 'navy', 'grey', 'crimson'], width=0.15, labelsAbove=False, saveName=None, dpi=300*)

makeBarplotCrossDatasets (*df_res_cross, figsize=(12, 7), c=['green', 'gold', 'navy', 'grey', 'crimson'], width=0.15, labelsAbove=False, saveName=None, dpi=300*)

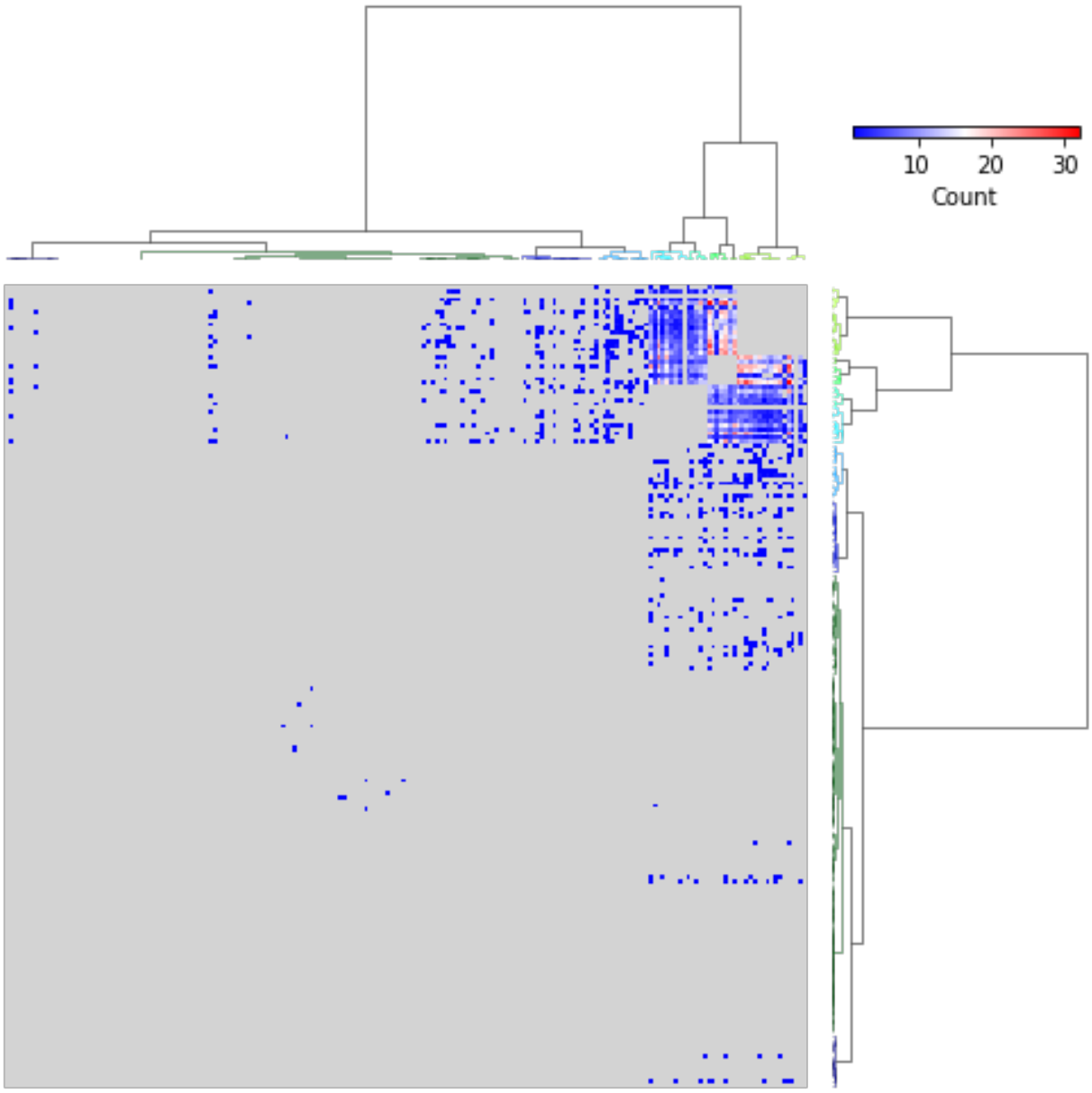
drawOneDownsampled (*usedf*, *ax*, *panel*, *a*, *b*, *c*, *v*, *loc='lower right'*, *xlabel='Training set size'*, *ylabel='Pearson corr. coef.'*, *col='val'*)

1.8.1 Plot Examples

plot_functions.**plotDendrogramWithKnownPairs**(*dfC*, *dfKS*)



plot_functions.**plotHeatmapPredictedSynergy**(*Z*, *pvalues*)


10
20
30
Count

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

a

INDEX