Reskit: a library for creating and curating reproducible pipelines for machine learning

Dmitry Petrov [1 2], Alexander Ivanov [2 4], Daniel Moyer [1],
Mikhail Belyaev[2 4] and Paul Thompson[1]

July 12
2017

Outline

— What is Reskit

— How it works

— Main features

— Reskit applications

— Limitations

— Further development

— Conclusion

What is Reskit

— A library for creating and curating reproducible pipelines for scientific machine learning.

— Main features include data caching, compatibility with most of the scikit-learn objects, optimization constraints such as forbidden combinations, and table generation for quality metrics.

github.com/neuro-ml/reskit

reskit.readthedocs.io

Reskit: roots

— Reskit heavily relies on scikit-learn architecture. Its core object is an extension of scikit-learn pipelines.

— It works with sklearn-like data transformation objects (with .fit, .transform and .fit_transform methods)

— It works with sklearn-like predictive modelling objects (with .fit and .predict methods)

— It also relies on pandas and (optionally) NetworkX and igraph

How Reskit works: toy problem

Imagine you want to perform the following **steps**: data scaling, dimensionality reduction and the fit classifier on your data. For each step you have several choices

— **scalers**: standard and min-max

— **dimensionality reduction**: PCA and kernel PCA

— **predictive models**: Logistic Regression and Decision Trees

Also, you don’t want to perform some steps with others. For example, you don’t want to run min-max scaling kernel PCA (toy example)

How Reskit works: defining steps

```
scalers = [('standard', StandardScaler()),
           ('minmax', MinMaxScaler())]
dim_reduction = [('pca', PCA()),
                 ('k_pca', KernelPCA())]
classifiers = [('LR', LogisticRegression()),
               ('DT', DecisionTreeClassifier())]
steps = [('scaler', scalers),
         ('dim_reduction', dim_reduction),
         ('classifier', classifiers)]
param_grid = {'LR': {'penalty': ['l1', 'l2']},
              'DT': {'max_depth': [2, 3, 7, 5]}}
banned_combos = [('minmax', 'k_pca')]
grid_cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=0)
eval_cv = ShuffleSplit(n_splits=5, shuffle=True, random_state=1)
```

How Reskit works: experiments plan

```
pipeliner = Pipeliner(steps, grid_cv=grid_cv, eval_cv=eval_cv, param_grid=param_grid,
banned_combos=banned_combos)
pipeliner.plan_table
```

	scaler	dim_reduction	classifier
0	standard	pca	LR
1	standard	pca	DT
2	standard	k_pca	LR
3	standard	k_pca	DT
4	minmax	pca	LR
5	minmax	pca	DT

How Reskit works: getting results

```
X, y = make_classification()
results = pipeliner.get_results(X, y, scoring=['roc_auc'])
results
```

scaler	dim_reduction	classifier	grid_roc_auc_mean	grid_roc_auc_std	grid_roc_auc_best_params	eval_roc_auc_mean	eval_roc_auc_std	eval_roc_auc_scores
standard	pca	LR	0.426	0.137928	{'penalty': 'l1'}	0.286	0.121754	[0.19 0.26 0.45 0.13 0.4]
standard	pca	DT	0.492	0.106141	{'max_depth': 5}	0.412	0.147973	[0.68 0.32 0.46 0.27 0.33]
standard	k_pca	LR	0.426	0.137928	{'penalty': 'l1'}	0.286	0.121754	[0.19 0.26 0.45 0.13 0.4]
standard	k_pca	DT	0.506	0.141046	{'max_depth': 3}	0.476	0.0796492	[0.61 0.365 0.465 0.495 0.445]
minmax	pca	LR	0.446	0.0722772	{'penalty': 'l1'}	0.334	0.0786384	[0.28 0.3 0.49 0.29 0.31]
minmax	pca	DT	0.541	0.118127	{'max_depth': 7}	0.472	0.117201	[0.55 0.45 0.6 0.5 0.26]

Reskit: features

— Ability to combine pipelines with an equal number of steps in list of experiments, running them and returning results in Pandas dataframe.

— Step caching. Reskit includes the option to save fixed steps, so in next pipeline specified steps won’t be recalculated.

— Forbidden combination constraints. You can block the unnecessary pairs combinations from experiments.

Reskit: features

— Evaluation of multiple performance metrics.

— DataTransformer class, which is Reskit's simplfied interface for specifying fit/transform methods in pipeline steps. A DataTransformer subclass need only specify one function.

— Tools for learning on graphs. Due to our original motivations Reskit includes a number of operations for network data. These were implemented using DataTransformer and in some cases the BCTpy (the Brain Connectivity Toolbox python version)

Applications: background

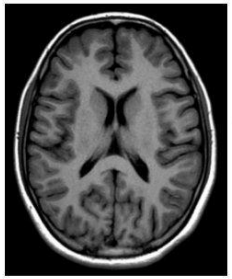
— We work with brain networks (connectomes) which are undirected graphs

— One subject can have several networks: different measurements, different ways of obtaining these networks

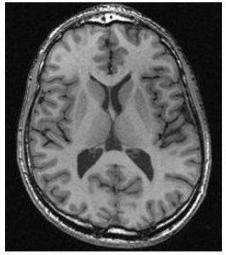
— Application 1: classify brain networks by diagnostic groups (i.e. healthy/ASD subjects)

— Application 2: compare different algorithms of building these networks using ICC, pairwise and gender classification

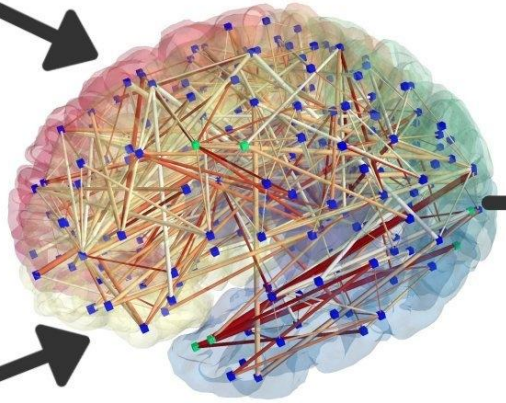
ML on networks: problem setting



T1W

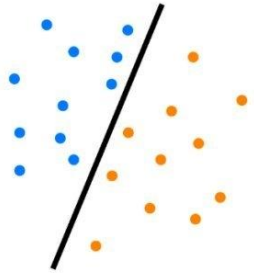


DWI



Brain network (connectome)

Network features



Classification

ML on networks: experiment steps

To classify brain networks you need to perform following steps:

— load adjacency matrices according to classification problem

— normalize matrices one way

— normalize matrices another way

— construct networks features

— scale/not scale features

— fit predictive models

https://github.com/neuro-ml/PRNI2016 (old version of Reskit)

ML on networks: reskit steps

```
data = [('UCLAsource', Transformer(get_autism)),
        ('UCLAbaseline', Transformer(get_baseline))]
weighters = [('origW', Transformer(orig)),
             ('binar', Transformer(binar_norm)),
             ('wbysqdist', Transformer(wbysqdist))]
normalizers = [('origN', Transformer(orig)),
               ('spectral', Transformer(spectral_norm))]
featurizers = [('origF', Transformer(orig, collect=['X'])),
               ('degrees', Transformer(degrees, collect=['degrees']))]
selectors = [('var_threshold', VarianceThreshold())]
scalers = [('minmax', MinMaxScaler()),
           ('origS', FunctionTransformer(orig))]
classifiers = [('LR', LogisticRegression()),
               ('RF', RandomForestClassifier()),
               ('SVC', SVC()),
               ('XGB', XGBClassifier(nthread=1)),
               ('SGD', SGDClassifier())]
```

https://github.com/neuro-ml/PRNI2016 (old version of Reskit)

ML on networks: reskit results sample

	Data	Weighters	Normalizers	Featurizers	Selectors	Scalers	Classifiers	grid_roc_auc_mean	grid_roc_auc_std	grid_roc_auc_best_param
0	UCLAsource	origW	spectral	degrees	var_threshold	minmax	LR	0.654255319149	0.20893086183	{'penalty': 'l1', 'max_iter': 50,
1	UCLAsource	origW	spectral	degrees	var_threshold	minmax	SVC	0.681914893617	0.177100023402	{'kernel': 'rbf', 'max_iter': 50,
2	UCLAsource	origW	spectral	degrees	var_threshold	minmax	SGD	0.682269503546	0.16890357928	{'penalty': 'elasticnet', 'alpha'
3	UCLAsource	origW	spectral	degrees	var_threshold	origS	RF	0.706737588652	0.204444110938	{'n_estimators': 500, 'criterio
4	UCLAsource	origW	spectral	degrees	var_threshold	origS	XGB	0.762677304965	0.173007607563	{'subsample': 1, 'reg_alpha':
5	UCLAsource	binar	spectral	degrees	var_threshold	minmax	LR	0.614539007092	0.0914240516795	{'penalty': 'l1', 'max_iter': 50,
6	UCLAsource	binar	spectral	degrees	var_threshold	minmax	SVC	0.357269503546	0.206535626934	{'kernel': 'linear', 'max_iter': 5
7	UCLAsource	binar	spectral	degrees	var_threshold	minmax	SGD	0.638120567376	0.146136662059	{'penalty': 'elasticnet', 'alpha'
8	UCLAsource	binar	spectral	degrees	var_threshold	origS	RF	0.609840425532	0.263017930915	{'n_estimators': 10, 'criterion'
9	UCLAsource	binar	spectral	degrees	var_threshold	origS	XGB	0.694503546099	0.16610522986	{'subsample': 0.7, 'reg_alpha
10	UCLAsource	wbysqdist	spectral	degrees	var_threshold	minmax	LR	0.75780141844	0.228233297542	{'penalty': 'l2', 'max_iter': 50,

https://github.com/neuro-ml/PRNI2016 (old version of Reskit)

Pairwise classification: problem setting

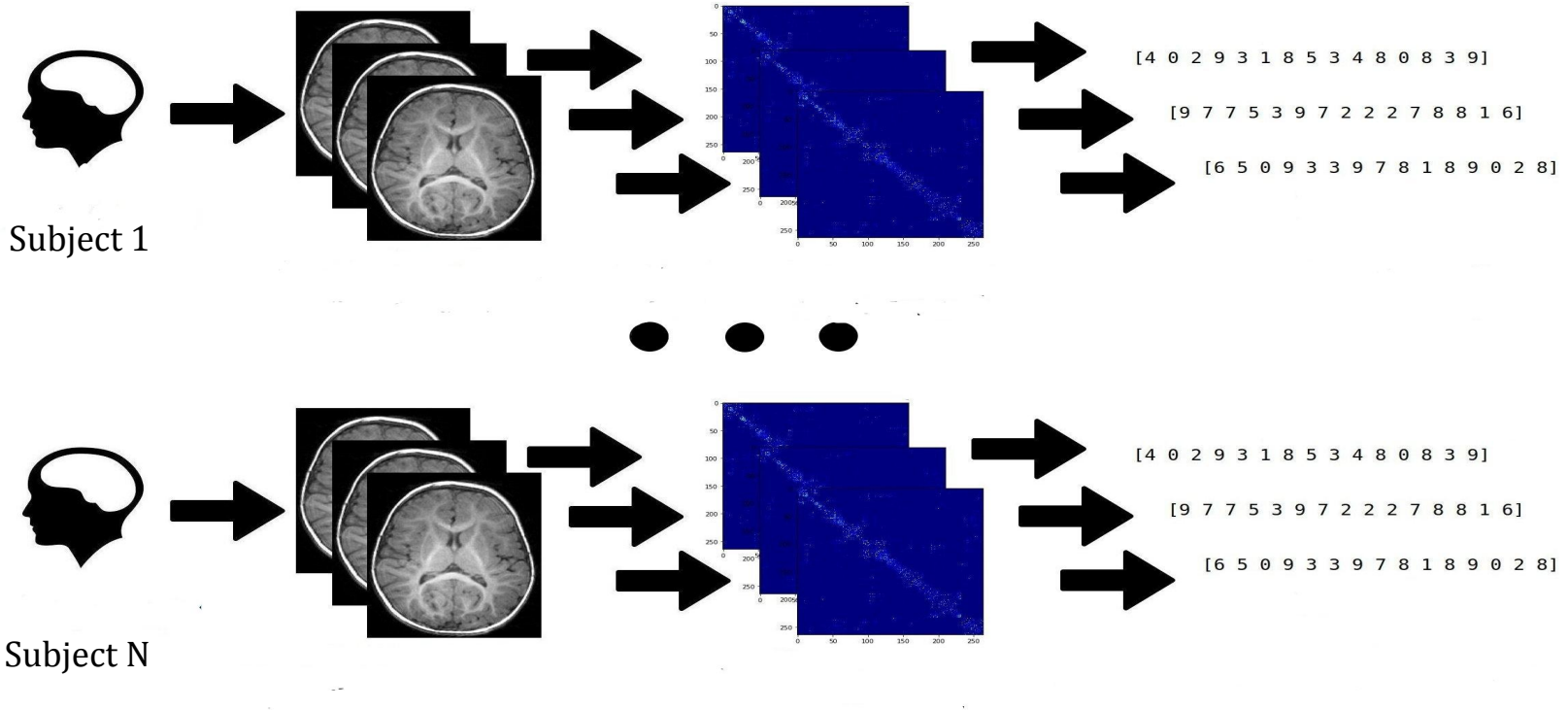
— To compare different connectome construction algorithms we used pairwise classification — discriminating pairs of connectomes as belonging to same individuals or not

— In addition to we calculated gender classification on the same data and Intraclass Correlation Coefficient (ICC)

— Connectomes were built without Reskit, we used it only for ML on ready-made datasets

https://github.com/lodurality/35_methods_MICCAI_2017

Pairwise classification: problem setting



Pairwise classification: problem setting

— Let's assume we have set of connectomes C_j^i, where where i-indices correspond to images and j-indices correspond to subjects and feature mapping $f : C \to \mathbb{R}^d$

— For each pair of connectome feature vectors $(f(C_{j_1}^{i_1}), f(C_{j_2}^{i_2}))$ we assign target variable 1 if they are from the same subject, 0 — else;

— We construct three pairwise differences of these vectors according to l_1, l_2 and l_∞ norms

Pairwise classification: challenge

Huge number of experiments (26640):

— 35 connectome building methods and 7 different network scales (parcellations)

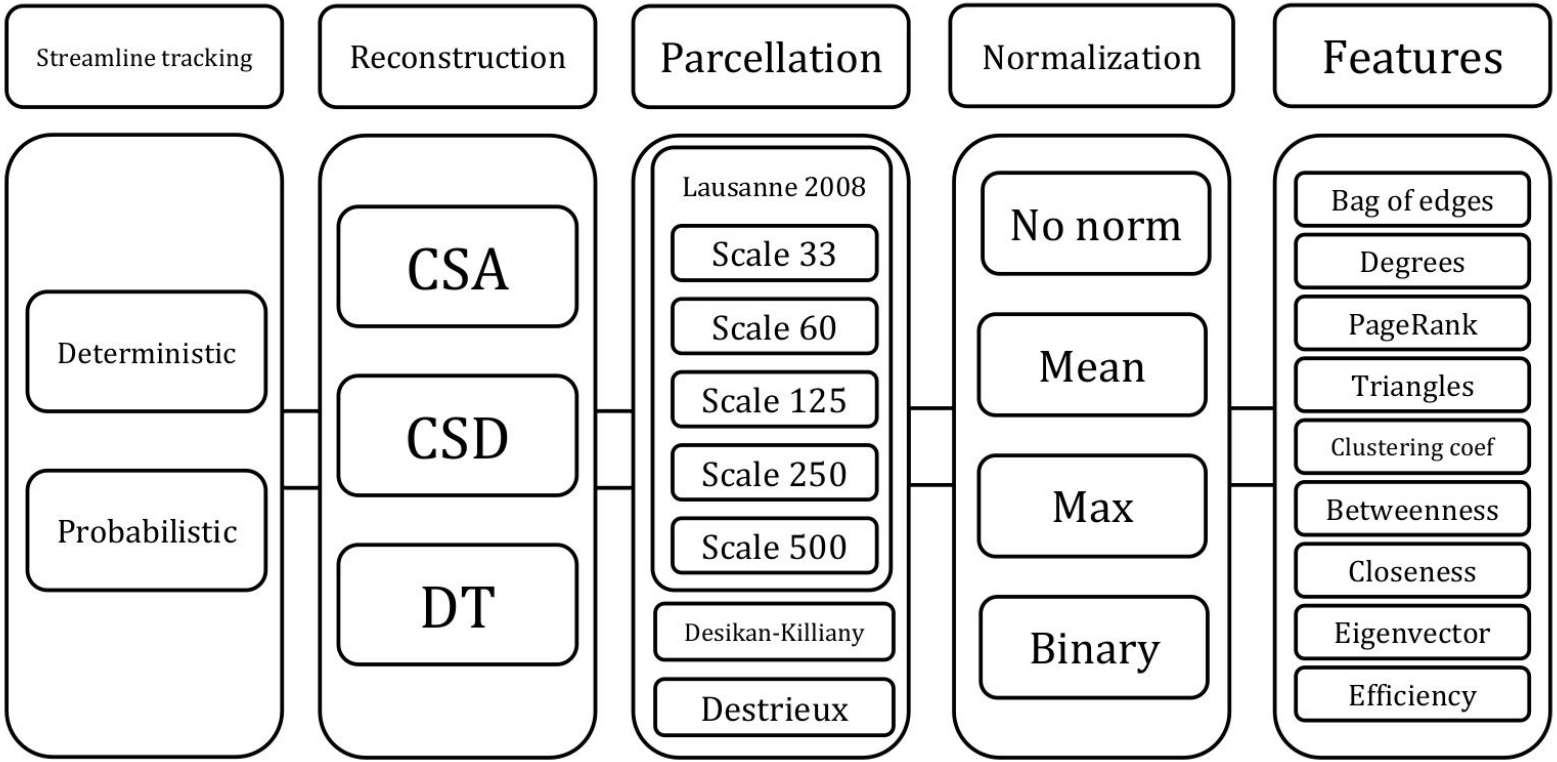
— 4 normalizations and 9 types of connectome features

— 3 different datasets

— for each combination we needed to do pairwise and gender classification amd ICC calculation

https://github.com/lodurality/35_methods_MICCAI_2017

Pairwise classification: challenge



https://github.com/lodurality/35_methods_MICCAI_2017

Pairwise classification: reskit steps

```
normalizers = [
        ('binar',   MatrixNormalizer(binar_norm)),
        ('max',     MatrixNormalizer(max_norm)),
        ('mean',    MatrixNormalizer(mean_norm)),
        ('no_norm', MatrixNormalizer(no_norm))
    ]
featurizers = [
        ('bag_of_edges',          MatrixFeaturizer([bag_of_edges])),
        ('degrees',               MatrixFeaturizer([degrees])),
        ('closeness_centrality',  MatrixFeaturizer([closeness_centrality])),
        ('betweenness_centrality',MatrixFeaturizer([betweenness_centrality])),
        ('eigenvector_centrality',MatrixFeaturizer([eigenvector_centrality])),
        ('pagerank',              MatrixFeaturizer([pagerank])),
        ('efficiency',            MatrixFeaturizer([efficiency])),
        ('clustering_coefficient',MatrixFeaturizer([clustering_coefficient])),
        ('triangles',             MatrixFeaturizer([triangles]))
    ]

pairwise_features = [
        ('l1_l2_linf', VectorFeaturizer(func_list=func_list))
    ]
```

https://github.com/lodurality/35_methods_MICCAI_2017

Pairwise classification: reskit results example

	Dataset	Normalizer	Featurizer	Pairwise_features	Scaler	Classifier	grid_accuracy_mean	grid_accuracy_st
0	HNU_1_deter_csa_con_ROIv_scale250	binar	bag_of_edges	l1_l2_linf	standard	LR	0.971851851852	0.0073516419412
1	HNU_1_deter_csa_con_ROIv_scale250	binar	bag_of_edges	l1_l2_linf	standard	SGD	0.972222222222	0.0070272836892
2	HNU_1_deter_csa_con_ROIv_scale250	binar	degrees	l1_l2_linf	standard	LR	0.995555555556	0.0034346735168
3	HNU_1_deter_csa_con_ROIv_scale250	binar	degrees	l1_l2_linf	standard	SGD	0.995555555556	0.0034346735168
4	HNU_1_deter_csa_con_ROIv_scale250	binar	closeness_centrality	l1_l2_linf	standard	LR	0.827777777778	0.0121151312843
5	HNU_1_deter_csa_con_ROIv_scale250	binar	closeness_centrality	l1_l2_linf	standard	SGD	0.801481481481	0.0105669204575
6	HNU_1_deter_csa_con_ROIv_scale250	binar	betweenness_centrality	l1_l2_linf	standard	LR	0.519259259259	0.0124281198041
7	HNU_1_deter_csa_con_ROIv_scale250	binar	betweenness_centrality	l1_l2_linf	standard	SGD	0.519259259259	0.0124281198041
8	HNU_1_deter_csa_con_ROIv_scale250	binar	eigenvector_centrality	l1_l2_linf	standard	LR	0.997407407407	0.0009072184232
9	HNU_1_deter_csa_con_ROIv_scale250	binar	eigenvector_centrality	l1_l2_linf	standard	SGD	0.997407407407	0.0009072184232

https://github.com/lodurality/35_methods_MICCAI_2017

Limitations

— Inability to provide custom grid search object (fixing)

— Inability to perform nested cross-validation (fixing)

— Inability to calculate statistics on data (fixing)

— Non-parallelizable

— Very narrow grid_cv/eval_cv experiment scheme

— Inability to save not only plans but experiments objects

— Inability to extend and combine experiment plans

Further development

— Ability to merge multiple experiment plans.

— Experiment customization (functions of pipelines)

— Distributed computing for calculation on clusters.

— Ability to calculate different quality metrics after one optimization.

— Collection of DataTransformers for various purposes.

— Option to save best models/pipelines according to external criteria.

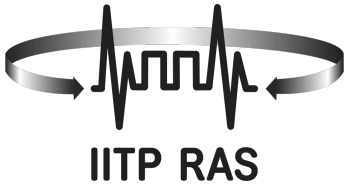
Conclusion: when Reskit may be useful for you

— You have a lot of data transformation/feature generation steps and want to compare them

— You want compare different models on different sets of features

— You want experiment results in convenient form for analysis (pandas dataframe)

Acknowledgements



Funding: NIH U54 EB020403 (ENIGMA CENTER FOR WORLDWIDE MEDICINE, IMAGING & GENOMICS), PI Paul M. Thompson; Russian Science Foundation (project 14-50-00150)

Thank you

Reskit: github.com/neuro-ml/reskit

Email: to.dmitry.petrov@gmail.com, alexander.radievich@gmail.com

Github: lodurality, hyperswitcher

Scipy 2017 Slack: @dmitry_petrov

✿**.. ╰╭✪‿✪╮╯ ..**✿