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$^1$

IGC
Imaging Genetics Center

IITP RAS

Skoltech
Skolkovo Institute of Science and Technology

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](https://github.com/neuro-ml/reskit)

[reskit.readthedocs.io](https://reskit.readthedocs.io)
— 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
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

```python
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

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

<table>
<thead>
<tr>
<th>scaler</th>
<th>dim_reduction</th>
<th>classifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>standard</td>
<td>pca</td>
</tr>
<tr>
<td>1</td>
<td>standard</td>
<td>pca</td>
</tr>
<tr>
<td>2</td>
<td>standard</td>
<td>k_pca</td>
</tr>
<tr>
<td>3</td>
<td>standard</td>
<td>k_pca</td>
</tr>
<tr>
<td>4</td>
<td>minmax</td>
<td>pca</td>
</tr>
<tr>
<td>5</td>
<td>minmax</td>
<td>pca</td>
</tr>
</tbody>
</table>
How Reskit works: getting results

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

<table>
<thead>
<tr>
<th>scaler</th>
<th>dim_reduction</th>
<th>classifier</th>
<th>grid_roc_auc_mean</th>
<th>grid_roc_auc_std</th>
<th>grid_roc_auc_best_params</th>
<th>eval_roc_auc_mean</th>
<th>eval_roc_auc_std</th>
<th>eval_roc_auc_scores</th>
</tr>
</thead>
<tbody>
<tr>
<td>standard</td>
<td>pca</td>
<td>LR</td>
<td>0.426</td>
<td>0.137928</td>
<td>{'penalty': 11}</td>
<td>0.206</td>
<td>0.121754</td>
<td>[0.19 0.26 0.45 0.13 0.4]</td>
</tr>
<tr>
<td>standard</td>
<td>pca</td>
<td>DT</td>
<td>0.492</td>
<td>0.106141</td>
<td>{'max_depth': 5}</td>
<td>0.412</td>
<td>0.147973</td>
<td>[0.68 0.32 0.46 0.27 0.33]</td>
</tr>
<tr>
<td>standard</td>
<td>k_pca</td>
<td>LR</td>
<td>0.426</td>
<td>0.137928</td>
<td>{'penalty': 11}</td>
<td>0.286</td>
<td>0.121754</td>
<td>[0.19 0.20 0.45 0.13 0.4]</td>
</tr>
<tr>
<td>standard</td>
<td>k_pca</td>
<td>DT</td>
<td>0.506</td>
<td>0.141046</td>
<td>{'max_depth': 3}</td>
<td>0.476</td>
<td>0.0796492</td>
<td>[0.61 0.365 0.465 0.495 0.445]</td>
</tr>
<tr>
<td>minmax</td>
<td>pca</td>
<td>LR</td>
<td>0.446</td>
<td>0.0722772</td>
<td>{'penalty': 11}</td>
<td>0.334</td>
<td>0.0786384</td>
<td>[0.28 0.3 0.49 0.29 0.31]</td>
</tr>
<tr>
<td>minmax</td>
<td>pca</td>
<td>DT</td>
<td>0.541</td>
<td>0.118127</td>
<td>{'max_depth': 7}</td>
<td>0.472</td>
<td>0.117201</td>
<td>[0.55 0.45 0.6 0.5 0.26]</td>
</tr>
</tbody>
</table>
```
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 simplified 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

Brain network (connectome)

T1W

DWI

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

```python
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

<table>
<thead>
<tr>
<th>Data</th>
<th>Weighters</th>
<th>Normalizers</th>
<th>Featurizers</th>
<th>Selectors</th>
<th>Scalers</th>
<th>Classifiers</th>
<th>grid_roc_auc_mean</th>
<th>grid_roc_auc_std</th>
<th>grid_roc_auc_best_params</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>UCLAsource</td>
<td>origW</td>
<td>spectral</td>
<td>degrees</td>
<td>var_threshold</td>
<td>minmax</td>
<td>0.654255319149</td>
<td>0.20893086183</td>
<td>{'penalty': 'l1', 'max_iter': 50,}</td>
</tr>
<tr>
<td>1</td>
<td>UCLAsource</td>
<td>origW</td>
<td>spectral</td>
<td>degrees</td>
<td>var_threshold</td>
<td>minmax</td>
<td>0.681914893617</td>
<td>0.17710023402</td>
<td>{'kernel': 'rbf', 'max_iter': 50,}</td>
</tr>
<tr>
<td>2</td>
<td>UCLAsource</td>
<td>origW</td>
<td>spectral</td>
<td>degrees</td>
<td>var_threshold</td>
<td>minmax</td>
<td>0.682269503546</td>
<td>0.16890357926</td>
<td>{'penalty': 'elasticnet', 'alpha':}</td>
</tr>
<tr>
<td>3</td>
<td>UCLAsource</td>
<td>origW</td>
<td>spectral</td>
<td>degrees</td>
<td>var_threshold</td>
<td>origS</td>
<td>0.706737588532</td>
<td>0.204444110938</td>
<td>{'n_estimators': 300, 'criterion':}</td>
</tr>
<tr>
<td>4</td>
<td>UCLAsource</td>
<td>origW</td>
<td>spectral</td>
<td>degrees</td>
<td>var_threshold</td>
<td>origS</td>
<td>0.762977304965</td>
<td>0.173007607563</td>
<td>{'subsample': 1, 'reg_alpha':}</td>
</tr>
<tr>
<td>5</td>
<td>UCLAsource</td>
<td>binar</td>
<td>spectral</td>
<td>degrees</td>
<td>var_threshold</td>
<td>minmax</td>
<td>0.614539007092</td>
<td>0.0914240516795</td>
<td>{'penalty': 'l1', 'max_iter': 50,}</td>
</tr>
<tr>
<td>6</td>
<td>UCLAsource</td>
<td>binar</td>
<td>spectral</td>
<td>degrees</td>
<td>var_threshold</td>
<td>minmax</td>
<td>0.357269503546</td>
<td>0.206535629394</td>
<td>{'kernel': 'linear', 'max_iter': 5}</td>
</tr>
<tr>
<td>7</td>
<td>UCLAsource</td>
<td>binar</td>
<td>spectral</td>
<td>degrees</td>
<td>var_threshold</td>
<td>minmax</td>
<td>0.638120567376</td>
<td>0.146136662059</td>
<td>{'penalty': 'elasticnet', 'alpha':}</td>
</tr>
<tr>
<td>8</td>
<td>UCLAsource</td>
<td>binar</td>
<td>spectral</td>
<td>degrees</td>
<td>var_threshold</td>
<td>origS</td>
<td>0.609840425532</td>
<td>0.263017930915</td>
<td>{'n_estimators': 10, 'criterion':}</td>
</tr>
<tr>
<td>9</td>
<td>UCLAsource</td>
<td>binar</td>
<td>spectral</td>
<td>degrees</td>
<td>var_threshold</td>
<td>origS</td>
<td>0.694503546099</td>
<td>0.16610522986</td>
<td>{'subsample': 0.7, 'reg_alpha':}</td>
</tr>
<tr>
<td>10</td>
<td>UCLAsource</td>
<td>wbysqdist</td>
<td>spectral</td>
<td>degrees</td>
<td>var_threshold</td>
<td>minmax</td>
<td>0.75780141844</td>
<td>0.228233297542</td>
<td>{'penalty': 'l2', 'max_iter': 50,}</td>
</tr>
</tbody>
</table>

[https://github.com/neuro-ml/PRNI2016](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

Subject 1

Subject N
Pairwise classification: problem setting

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

— For each pair of connectome feature vectors $(f(C^i_{j1}), f(C^i_{j2}))$ 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 $l_\infty$ 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

```python
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

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Normalizer</th>
<th>Featurizer</th>
<th>Pairwise_features</th>
<th>Scaler</th>
<th>Classifier</th>
<th>grid_accuracy_mean</th>
<th>grid_accuracy_std</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>binar</td>
<td>bag_of_edges</td>
<td>l1_l2_inf</td>
<td>standard</td>
<td>LR</td>
<td>0.971851851852</td>
<td>0.0073516419412</td>
</tr>
<tr>
<td>1</td>
<td>binar</td>
<td>bag_of_edges</td>
<td>l1_l2_inf</td>
<td>standard</td>
<td>SGD</td>
<td>0.972222222222</td>
<td>0.0070272636928</td>
</tr>
<tr>
<td>2</td>
<td>binar</td>
<td>degrees</td>
<td>l1_l2_inf</td>
<td>standard</td>
<td>LR</td>
<td>0.995555555556</td>
<td>0.0034346735168</td>
</tr>
<tr>
<td>3</td>
<td>binar</td>
<td>degrees</td>
<td>l1_l2_inf</td>
<td>standard</td>
<td>SGD</td>
<td>0.995555555556</td>
<td>0.0034346735168</td>
</tr>
<tr>
<td>4</td>
<td>binar</td>
<td>closeness_centrality</td>
<td>l1_l2_inf</td>
<td>standard</td>
<td>LR</td>
<td>0.827777777777</td>
<td>0.0121151312343</td>
</tr>
<tr>
<td>5</td>
<td>binar</td>
<td>closeness_centrality</td>
<td>l1_l2_inf</td>
<td>standard</td>
<td>SGD</td>
<td>0.801481481481</td>
<td>0.0106669204575</td>
</tr>
<tr>
<td>6</td>
<td>binar</td>
<td>betweenness_centrality</td>
<td>l1_l2_inf</td>
<td>standard</td>
<td>LR</td>
<td>0.519259259259</td>
<td>0.0124281198041</td>
</tr>
<tr>
<td>7</td>
<td>binar</td>
<td>betweenness_centrality</td>
<td>l1_l2_inf</td>
<td>standard</td>
<td>SGD</td>
<td>0.519259259259</td>
<td>0.0124281198041</td>
</tr>
<tr>
<td>8</td>
<td>binar</td>
<td>eigenvector_centrality</td>
<td>l1_l2_inf</td>
<td>standard</td>
<td>LR</td>
<td>0.997407407407</td>
<td>0.0009072184232</td>
</tr>
<tr>
<td>9</td>
<td>binar</td>
<td>eigenvector_centrality</td>
<td>l1_l2_inf</td>
<td>standard</td>
<td>SGD</td>
<td>0.997407407407</td>
<td>0.0009072184232</td>
</tr>
</tbody>
</table>

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

✿*
∗°´¯ ꒪ ꒳ ꒳ ꒳ ꒳ ꒳ ꒳ ꒳ ꒳ °´¯ ✿