Classifier Training & Optimization. Reproducible way.

Tatiana Likhomanenko, Alexey Rogozhnikov,
Andrey Ustyuzhanin, Alexander Baranov, Egor Khairullin

IML 2016-02-03

Irreproducibility indicators

- ‘Which version of my code I used to generate figure 13?’
- ‘The new student wants to reuse that model I published three years ago but he can’t reproduce the figures’
- ‘I thought I used the same parameters but I’m getting different results…’
- ‘On what dataset have I compared algorithms exactly?’
- ‘It worked yesterday!!’
- ‘Why did I do that?!’

> At last the wall is complete,
> rock-solid! no single crack!
> but… where is the exit?

Unknown Japanese poet

Reproducible Experiment Platform

- Python-based (numpy, pandas, …), Jupyter-friendly
- Unified scikit-learn-like API to many ML packages
  (Sklearn, XGBoost, uBoost, TMVA, Theanets, …)
- Meta-algorithms pipelines («REP lego»)
- Configurable interactive reporting & visualization
  to ensure model quality (e.g. check for overfitting)
- Pluggable quality metrics
- Parallelized training of classifiers & grid search (IPython parallel)
- Demo server: https://lhcb-rep.cern.ch, password: ‘`rep`’
- Github: https://github.com/yandex/rep

Unified classifier interface

jupyter 01-howto-Classifiers (autosaved)

File Edit View Insert Cell Kernel Help | Python 2

Markdown | Cell Toolbar: None

Classifiers

All classifiers inherit from **sklearn.BaseEstimator** and have the following methods:

- `classifier.fit(X, y, sample_weight=None)` - train classifier
- `classifier.predict_proba(X)` - return probabilities vector for all classes
- `classifier.predict(X)` - return predicted labels
- `classifier.staged_predict_proba(X)` - return probabilities after each iteration (not supported by TMVA)
- `classifier.get_feature_importances()`

Here we use `X` to denote matrix with data of shape `[n_samples, n_features]`, `y` is vector with labels (0 or 1) of shape [n_samples], `sample_weight` is vector with weights.

Difference from default scikit-learn interface

X should be* `pandas.DataFrame`, not `numpy.array`.
Provided this, you'll be able to choose features used in training by setting e.g. `features=['FlightTime', 'p']` in constructor.

* it works fine with `numpy.array` as well, but in this case all the features will be used.

https://github.com/yandex/rep/blob/master/howto/01-howto-Classifiers.ipynb

Meta Machine Learning (REP-Lego)

1. Factory
2. Grid Search
    a. GridOptimalSearch
    b. Folding Scorer
    c. Various Optimization algorithms
3. Interface of parameter optimizer
4. Folding
   https://github.com/yandex/rep/blob/master/howto/04-howto-folding.ipynb
5. Stacking

REP-Lego

jupyter 01-howto-Classifiers (autosaved)

File Edit View Insert Cell Kernel Help | Python 2

Markdown | Cell Toolbar: None

AdaBoost over XGBoost

In [21]:

```
from sklearn.ensemble import AdaBoostClassifier
```

In [22]:

```
%%time
base_xgb = XGBoostClassifier(n_estimators=20)
ada_xgb = SklearnClassifier(AdaBoostClassifier(base_estimator=base_xgb, n_estimators=5))
ada_xgb.fit(train_data[variables], train_labels)
print('training complete!')

# predict probabilities for each class
prob = ada_xgb.predict_proba(test_data[variables])
print 'AUC', roc_auc_score(test_labels, prob[:, 1])

# predict probabilities for each class
prob = ada_xgb.predict_proba(train_data[variables])
print 'AUC', roc_auc_score(train_labels, prob[:, 1])
```

```
training complete!
AUC 0.975709190087
AUC 0.998466758443
CPU times: user 34 s, sys: 266 ms, total: 34.3 s
Wall time: 34.4 s
```

https://github.com/yandex/rep/blob/master/howto/01-howto-Classifiers.ipynb

REP-Lego. TMVA

jupyter 01-howto-Classifiers (autosaved)

File Edit View Insert Cell Kernel Help | Python 2

Markdown | Cell Toolbar: None

AdaBoost over TMVA classifier

the following code shows that you can do the same with i.e. TMVA, uncomment it to try

In [23]:
```
# base_tmva = TMVAClassifier(method='kBDT', NTrees=20)
# ada_tmva  = SklearnClassifier(AdaBoostClassifier(base_estimator=base_tmva, n_estimators=5), features=variables)
# ada_tmva.fit(train_data, train_labels)
# print('training complete')

# prob = ada_tmva.predict_proba(test_data)
# print 'AUC', roc_auc_score(test_labels, prob[:, 1])
```

Other advantages of common interface

There are many things you can do with classifiers now:

- cloning
- getting / setting parameters as dictionaries
- automatic hyperparameter optimization
- build pipelines (`sklearn.pipeline`)
- use hierarchical training, training on subsets
- passing over internet / train classifiers on other machines

And you can replace classifiers at any moment.

https://github.com/yandex/rep/blob/master/howto/01-howto-Classifiers.ipynb

REP-Lego. Classifier Factories

REP (Reproducible Experiment Platform)

Search docs

- Data
- Estimators (classification and regression)
- Meta Machine Learning
  - Factory
  - Factory Examples
  - Grid Search
  - Folding
  - Stacking
- Report for models
- Plotting
- Utilities
- Howto notebooks

Factory Examples

- Prepare dataset

```
>>> from sklearn import datasets
>>> import pandas, numpy
>>> from rep.utils import train_test_split
>>> from sklearn.metrics import roc_auc_score
>>> # iris data
>>> iris = datasets.load_iris()
>>> data = pandas.DataFrame(iris.data, columns=['a', 'b', 'c', 'd'])
>>> labels = iris.target
>>> # Take just two classes instead of three
>>> data = data[labels != 2]
>>> labels = labels[labels != 2]
>>> train_data, test_data, train_labels, test_labels = train_test_split(data, labels, tr
```

- Train factory of classifiers

```
>>> from rep.metaml import ClassifiersFactory
>>> from rep.estimators import TMVAClassifier, SklearnClassifier, XGBoostClassifier
>>> from sklearn.ensemble import GradientBoostingClassifier
>>> factory = ClassifiersFactory()
>>> estimators
>>> factory.add_classifier('tmva', TMVAClassifier(method='kBDT', NTrees=100, Shrinkage=0
>>> factory.add_classifier('ada', GradientBoostingClassifier())
>>> factory['xgb'] = XGBoostClassifier(features=['a', 'b'])
>>> factory.fit(train_data, train_labels)
model ef          was trained in 0.22 seconds
```

https://github.com/yandex/rep/blob/master/howto/02-howto-Factory.ipynb

Reporting

- Draws set of reports upon model training completion. Supported libraries:
  - Matplotlib
  - ROOT
  - Bokeh (Javascript)
  - plot.ly
- Extensible!

https://github.com/yandex/rep/blob/master/howto/02-howto-Factory.ipynb

Reporting

jupyter 02-howto-Factory (autosaved)

File Edit View Insert Cell Kernel Help | Python 2

Markdown | Cell Toolbar: None

Get ClassificationReport object

report has many useful methods

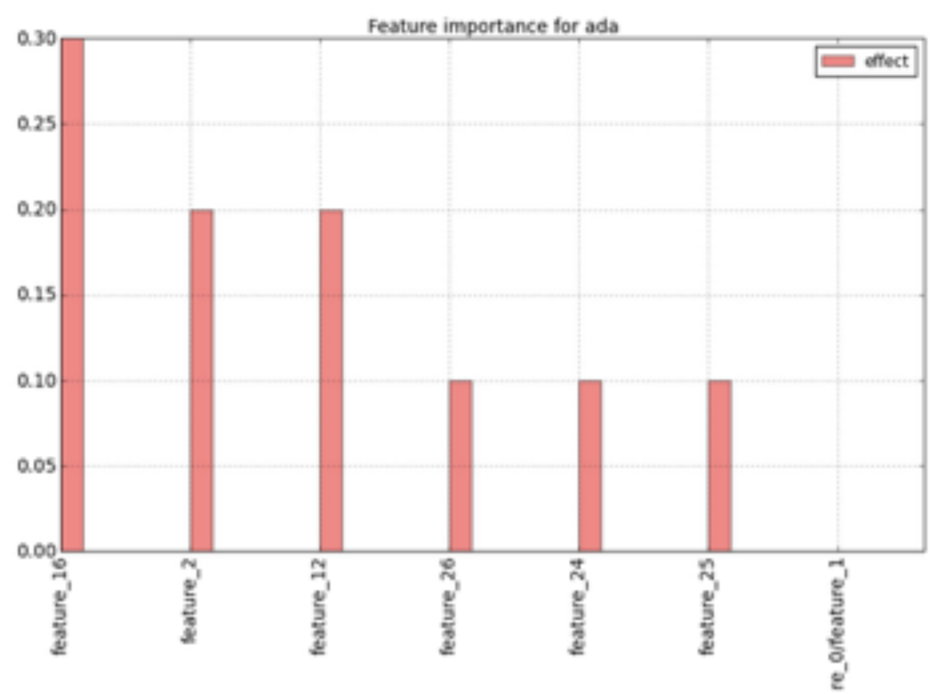
In [12]:
```
report = factory.test_on(test_data, test_labels)
```

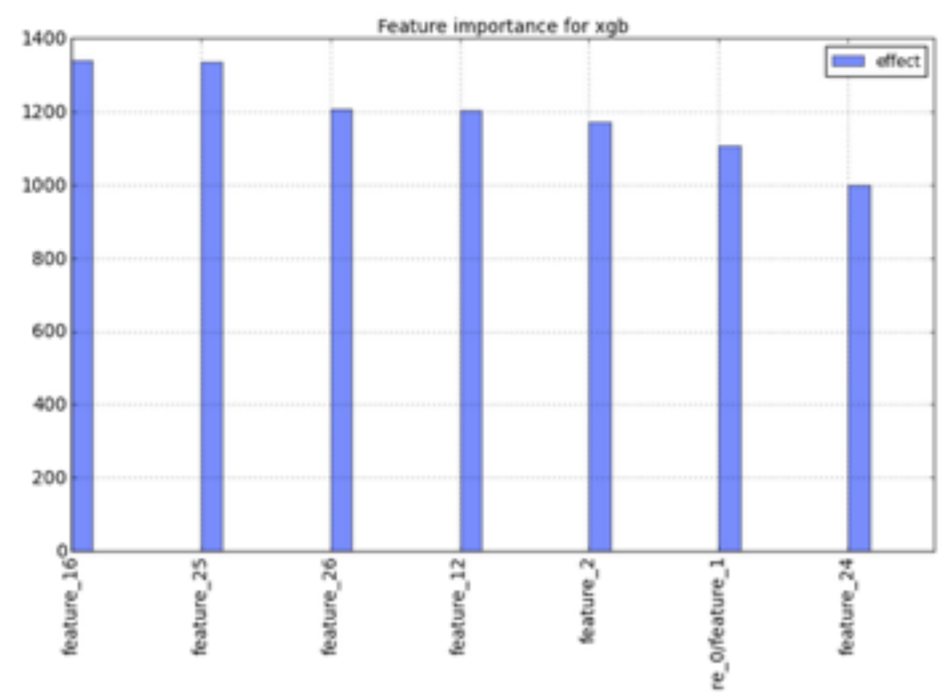
Plot importances of features

Only the features used in training are compared

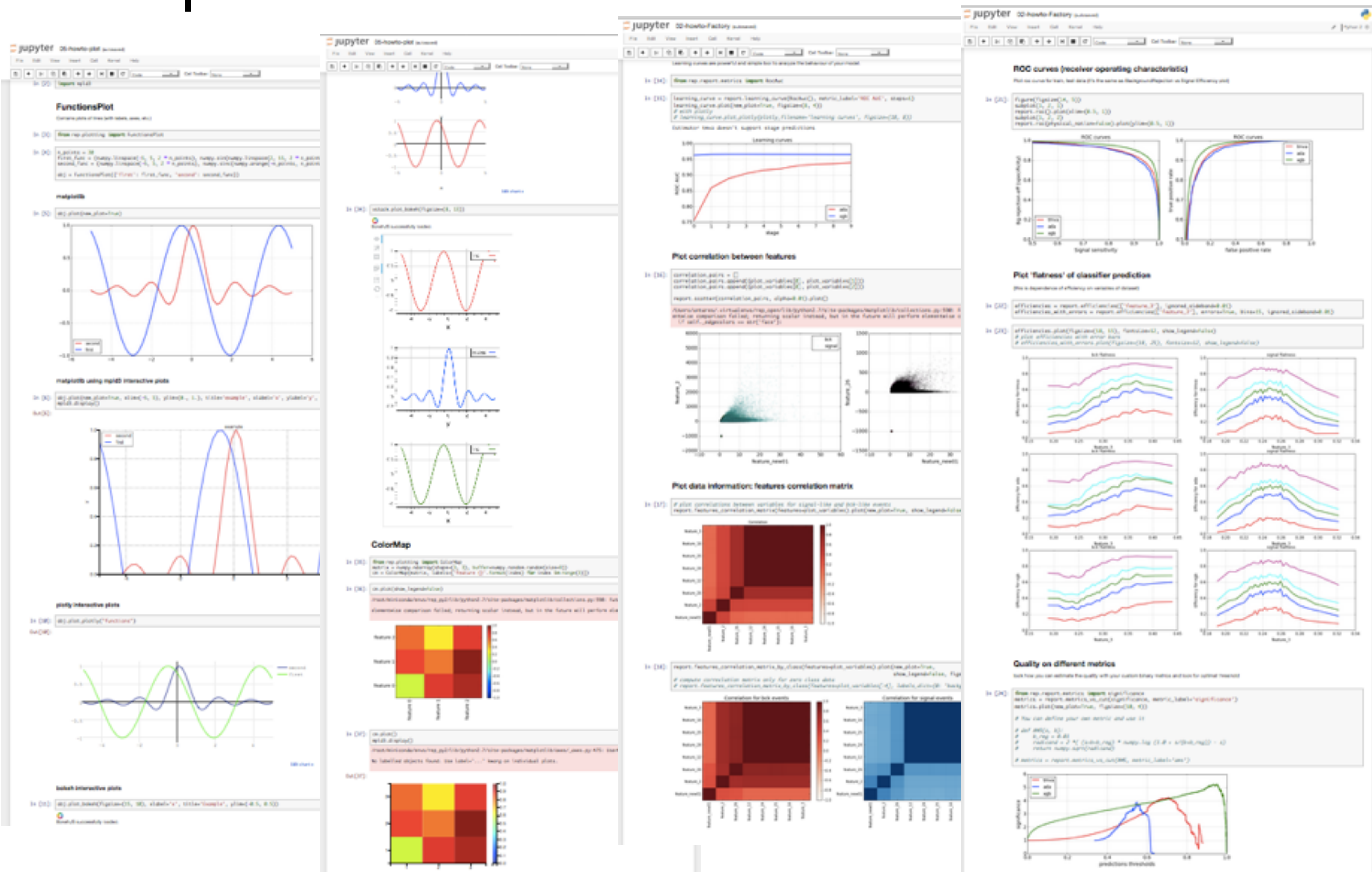
In [13]:
```
features_importances = report.feature_importance()
features_importances.plot()
# not only in matplotlib, but in other libraries too. For instance, with plotly
# features_importances.plot_plotly('importances', figsize=(15, 6))
```

```
Estimator tmva doesn't support feature importances
```





More plots


jupyter
FunctionsPlot
ColorMap
Plot correlation between features
Plot data information: features correlation matrix
ROC curves (receiver operating characteristic)
Plot 'flatness' of classifier prediction
Quality on different metrics

Metrics

Quality on different metrics

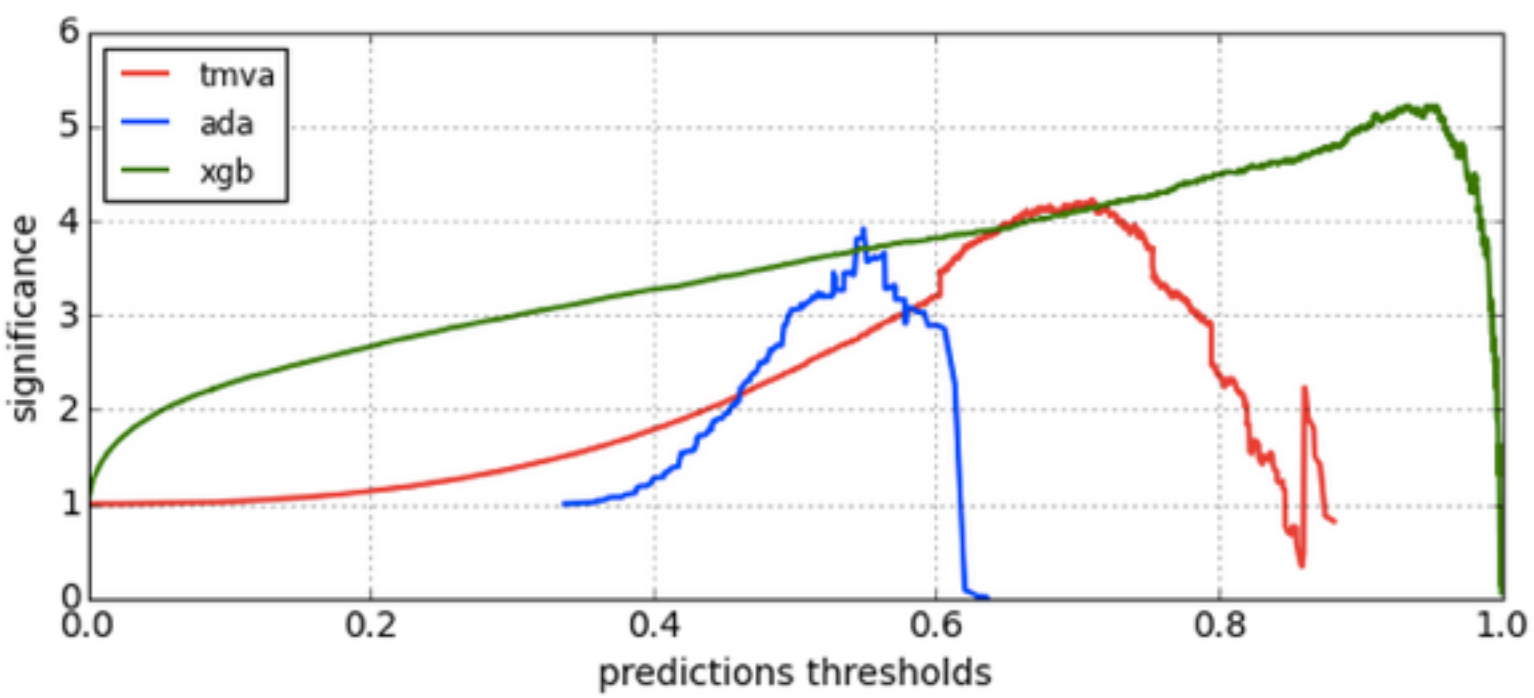
look how you can estimate the quality with your custom binary metrics and look for optimal threshold

In [24]:
```
from rep.report.metrics import significance
metrics = report.metrics_vs_cut(significance, metric_label='significance')
metrics.plot(new_plot=True, figsize=(10, 4))

# You can define your own metric and use it

# def AMS(s, b):
#     b_reg = 0.01
#     radicand = 2 *( (s+b+b_reg) * numpy.log (1.0 + s/(b+b_reg)) - s)
#     return numpy.sqrt(radicand)

# metrics = report.metrics_vs_cut(AMS, metric_label='ams')
```



Parallelized training & optimization

jupyter 02-howto-Factory (autosaved)

File Edit View Insert Cell Kernel Help | Python 2

Markdown Cell Toolbar: None

Factory of different models

This class is OrderedDict, with additional interface, main methods are:

- `factory.add_classifier(name, classifier)`
- `factory.fit(X, y, sample_weight=None, ipc_profile=None, features=None)`
  train all classifiers in factory
  if `features` is not None, then all classifiers will be trained on these features
  you can pass the name of ipython cluster via `ipc_profile` for parallel training
- `factory.test_on_lds(lds)` - test all models on lds(`rep.data.storage.LabeledDataStorage`)
  returns report (`rep.report.classification.ClassificationReport`)

In [6]:
```
from rep.metaml import ClassifiersFactory
from rep.estimators import TMVAClassifier, SklearnClassifier, XGBoostClassifier
from sklearn.ensemble import AdaBoostClassifier
```

Define classifiers (that will be compared)

Please pay attention that we set very small number of trees, just to make this notebook work fast. Don't forget to tune classifier!

In [7]:
```
factory = ClassifiersFactory()
# There are different ways to add classifiers to Factory:
factory.add_classifier('tmva', TMVAClassifier(NTrees=50, features=train_variables, Shrinkage=0.05))
factory.add_classifier('ada', AdaBoostClassifier(n_estimators=10))
factory['xgb'] = XGBoostClassifier(features=train_variables)
```

https://github.com/yandex/rep/blob/master/howto/02-howto-Factory.ipynb

Parallelized optimization. Grid Search.

```
In [4]: import numpy
        import numexpr
        import pandas
        from rep import utils
        from sklearn.ensemble import GradientBoostingClassifier
        from rep.report.metrics import RocAuc
        from rep.metaml import GridOptimalSearchCV, FoldingScorer, RandomParameterOptimizer
        from rep.estimators import SklearnClassifier, TMVAClassifier, XGBoostRegressor
```

```
In [5]: # define grid parameters
        grid_param = {}
        grid_param['learning_rate'] = [0.2, 0.1, 0.05, 0.02, 0.01]
        grid_param['max_depth'] = [2, 3, 4, 5]

        # use random hyperparameter optimization algorithm
        generator = RandomParameterOptimizer(grid_param)

        # define folding scorer
        scorer = FoldingScorer(RocAuc(), folds=3, fold_checks=3)
```

```
In [6]: %%time
        estimator = SklearnClassifier(GradientBoostingClassifier(n_estimators=30))
        grid_finder = GridOptimalSearchCV(estimator, generator, scorer, parallel_profile='threads-4')
        grid_finder.fit(data, labels)

        Performing grid search in 4 threads
        4 evaluations done
        8 evaluations done
        10 evaluations done
        CPU times: user 47.2 s, sys: 772 ms, total: 48 s
        Wall time: 17.5 s
```

https://github.com/yandex/rep/blob/master/howto/02-howto-gridsearch.ipynb

Running REP

- Locally
  - virtualenv, conda,
    https://github.com/yandex/rep/wiki/Installing-manually
  - docker, kitematic,
    https://github.com/yandex/rep/wiki/Install-REP-with-Docker-(Mac-OS-X,-Windows)
- CERN openstack
  https://github.com/yandex/rep/wiki/Install-REP-at-openstack
- Any cloud provider

Analysis reproducibility model

1. Install REP-server (e.g. at CERN openstack)
2. Give access to all your team members (ROOTaaS?)
3. integrate with github, gitlab repository
4. Automate building/execution
5. Add Dockerfile to the repository:

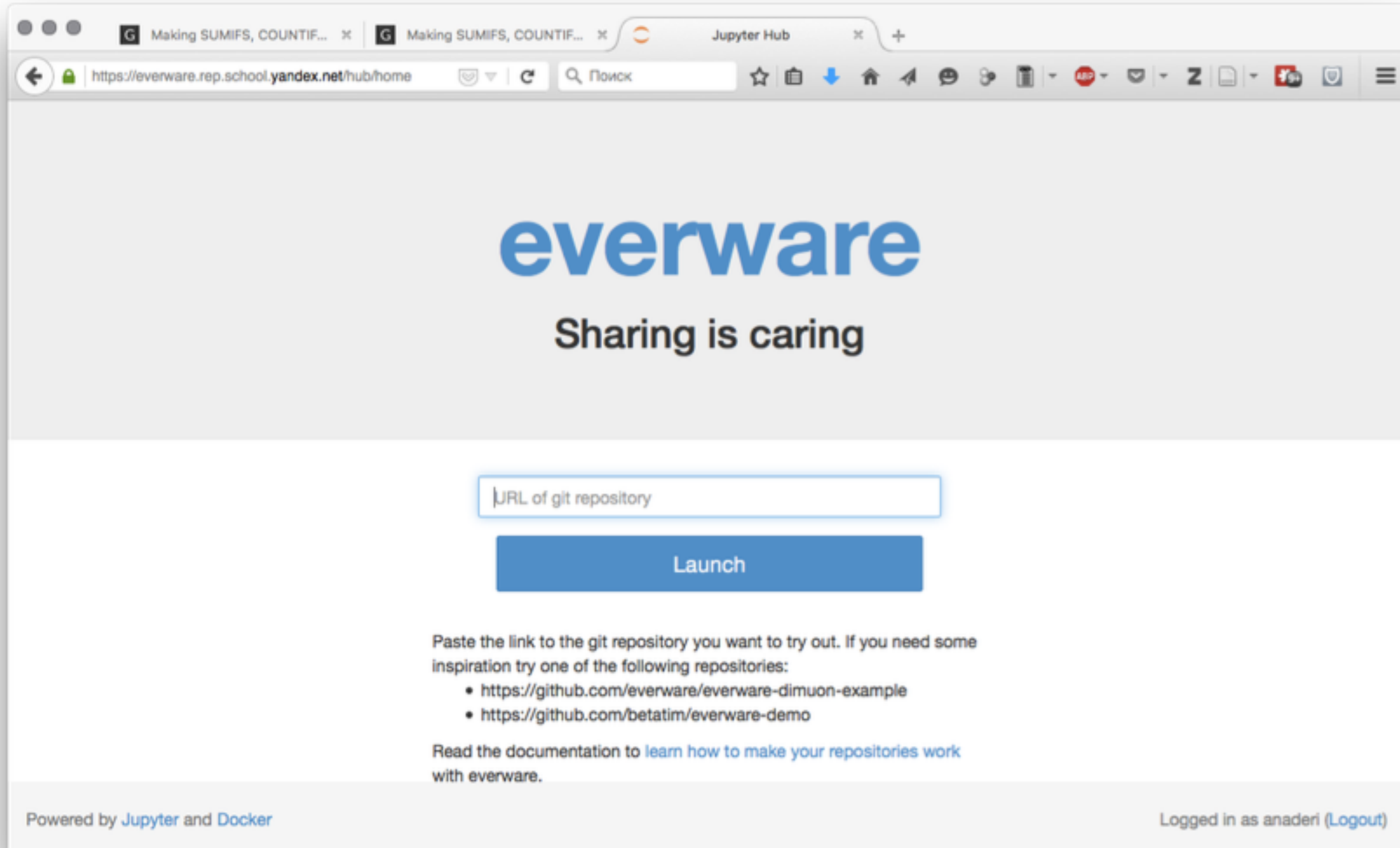
6 lines (4 sloc) | 144 Bytes — Raw | Blame | History

```
FROM yandex/rep:0.6.4
MAINTAINER Noel Dawe <noel.dawe@cern.ch>

# RUN bash --login -c "pip install rootpy==0.8.0"
# RUN apt-get install -y curl
```

6. You can rely on image version (e.g. **0.6.4**) + git versioning
7. Use everware

Analysis re-run interactively



http://everware.xyz

Everware, behind the scenes

- Docker (+swarm)
  - docker swarm spawner based on dockerspawner https://github.com/everware/dockerspawner
  - containers https://github.com/everware/container-tools
- JupyterHub
- GitHub
- Tools & docs to simplify
  - docker image creation
  - playing with analysis locally

REP for research & education

- High Energy Physics
  - online & offline data analysis at CERN
  - optimization of disk storage
- AstroPhysics
  - CRAYFIS (http://crayfis.io)
- Industry
  - Yandex Data Factory
- Education
  - MLHEP summer school (http://www.hse.ru/mlhep2015/)
  - Several hackathons

Conclusion

- REP is a serious effort to improve research reproducibility by a leading IT company (Yandex)
- Integrates with well-known ML libraries (sklearn, XGboost, Theanets, TMVA, ...)
- Works well along with git & docker for analysis preservation
- REP is open
  - provided under Apache-2.0 License
  - well-documented: http://yandex.github.io/rep/
  - flexible and extensible
  - and used in industry & scientific research & education
- Zurich workshop on HEP and ML, Feb 18, https://indico.cern.ch/event/433556/

Thank you!

Andrey Ustyuzhanin

anaderiRu@twitter

andrey.ustyuzhanin@cern.ch