Lokki Documentation

Version 0.0.1

July 2020

Table of Contents

About

Lokki is an open source automatic machine learning library providing novel visualization and pipeline identification techniques within the Python programming language.

Users also have the ability to output models for later use and analyze formatted 16s data sets or precomputed results.

Installation

Lokki can be installed on local computers using the python package index (PyPi)

```
pip install lokki
```

Data Format

Data input require a tabular format (.txt or .csv) and must adhere to one of two layouts. If performing a de novo search, users must provide an OTU table with column headers that match the table below. Note that the column names and format must match exactly.

Table 1a: OTU Table Format

Sample	OTU_1	OTU_2	…	OTU_N	Target
id_1	…	…	…	…	0
Id_2	…	…	…	…	1
…	…	…	…	…	…
id_m	…	…	…	…	0

If users want to analyze pre-computed performance, the user must construct binary features that indicate whether the feature of interest was present in that sample or not. The appropriate format is given below. The feature column names do not need to match the table below, but the “Sample” and “Score” column names must be present.

Table 1b: Precompute Results Format

Sample	Feature_1	Feature_2	…	Feature_N	Score
id_1	0	1	…	0	0.677
Id_2	1	1	…	0	0.786
…	…	…	…	…	…
id_m	0	1	…	1	0.976

A binary feature could be any characteristic of the data the user would like to analyze. For example, a binary feature could represent whether methylation data was used, where 1 would indicate presence and 0 would indicate absence.

Search Options

Lokki currently has 3 data transformation methods, 8 feature selection / feature engineering methods, and 10 machine learning algorithms. These are summarized in the table below giving the search name that the package will understand and a full description of the method.

Table 2a: Data transformation methods

Name	Description
none	No Data Transformation
log	Log Transformation
zscore	Z Score Transformation

Table 2b: Feature selection and feature engineering methods

Name	Description
none	No Feature Transformation
hfe	Hierarchical Feature Engineering
chi_square	Chi Square Feature Selection
mutual_information	Mutual Information Feature Selection
pca	Principal Component Analysis
factor_analysis	Factor Analysis
ica	Fast Independent Component Analysis
nma	Non-Negative Matrix Factorization

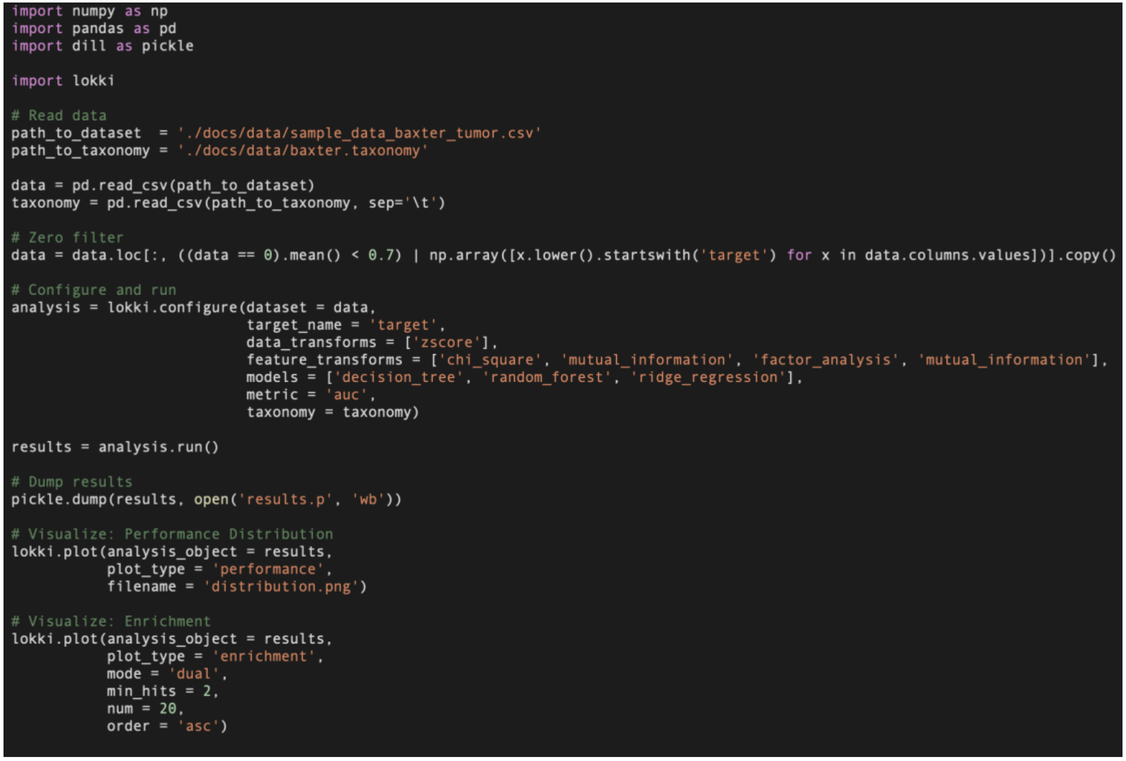
Table 2c: Machine learning algorithms

Name	Description
decision_tree	Decision Trees
random_forest	Random Forest
lda	Linear Discriminant Analysis
qda	Quadratic Discriminant Analysis
extra_tree	Extreme Randomized Trees
logistic_regression	Logistic Regression
adaboost	AdaBoost
gradient_boosting	Gradient Boosting
svm	Support Vector Machine
ridge_regression	Ridge Regression

Visualizations

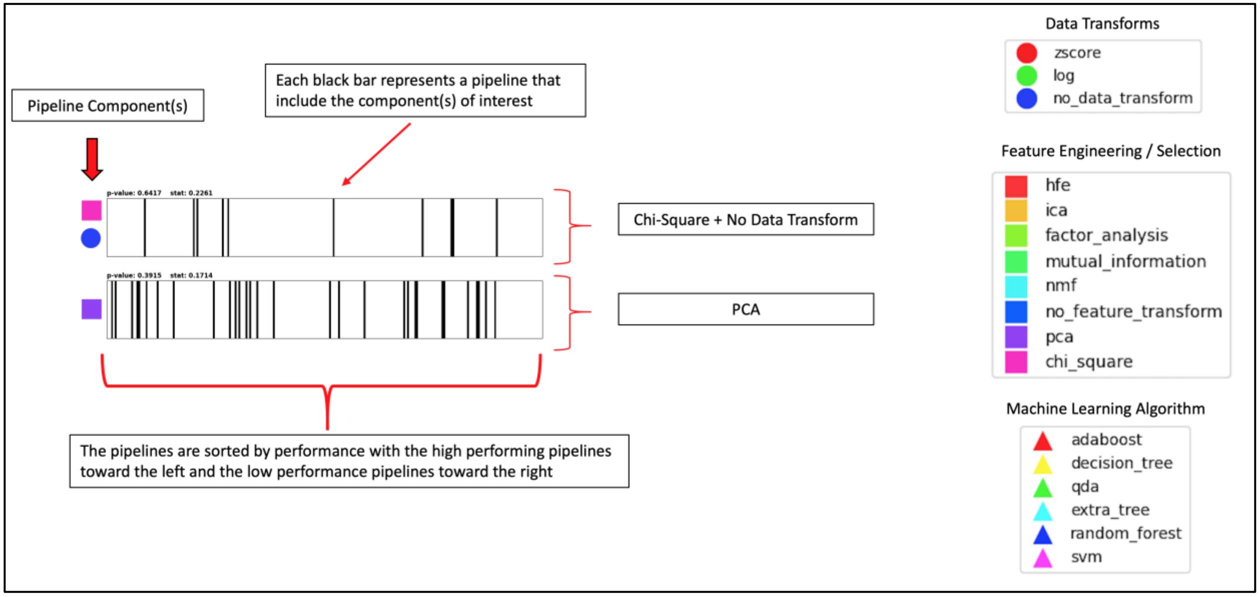
There are currently two methods of visualizing the data from a de novo search using an OTU input table: Performance Distribution and Enrichment Plot. The example code below shows how to prepare these visualizations followed by sample plots.

Figure 3a



```
import numpy as np
import pandas as pd
import dill as pickle

import lokki

# Read data
path_to_dataset  = './docs/data/sample_data_baxter_tumor.csv'
path_to_taxonomy = './docs/data/baxter.taxonomy'

data = pd.read_csv(path_to_dataset)
taxonomy = pd.read_csv(path_to_taxonomy, sep='\t')

# Zero filter
data = data.loc[:, ((data == 0).mean() < 0.7) | np.array([x.lower().startswith('target') for x in data.columns.values])].copy()

# Configure and run
analysis = lokki.configure(dataset = data,
                           target_name = 'target',
                           data_transforms = ['zscore'],
                           feature_transforms = ['chi_square', 'mutual_information', 'factor_analysis', 'mutual_information'],
                           models = ['decision_tree', 'random_forest', 'ridge_regression'],
                           metric = 'auc',
                           taxonomy = taxonomy)

results = analysis.run()

# Dump results
pickle.dump(results, open('results.p', 'wb'))

# Visualize: Performance Distribution
lokki.plot(analysis_object = results,
           plot_type = 'performance',
           filename = 'distribution.png')

# Visualize: Enrichment
lokki.plot(analysis_object = results,
           plot_type = 'enrichment',
           mode = 'dual',
           min_hits = 2,
           num = 20,
           order = 'asc')
```

Figure 3b



Pipeline Component(s)

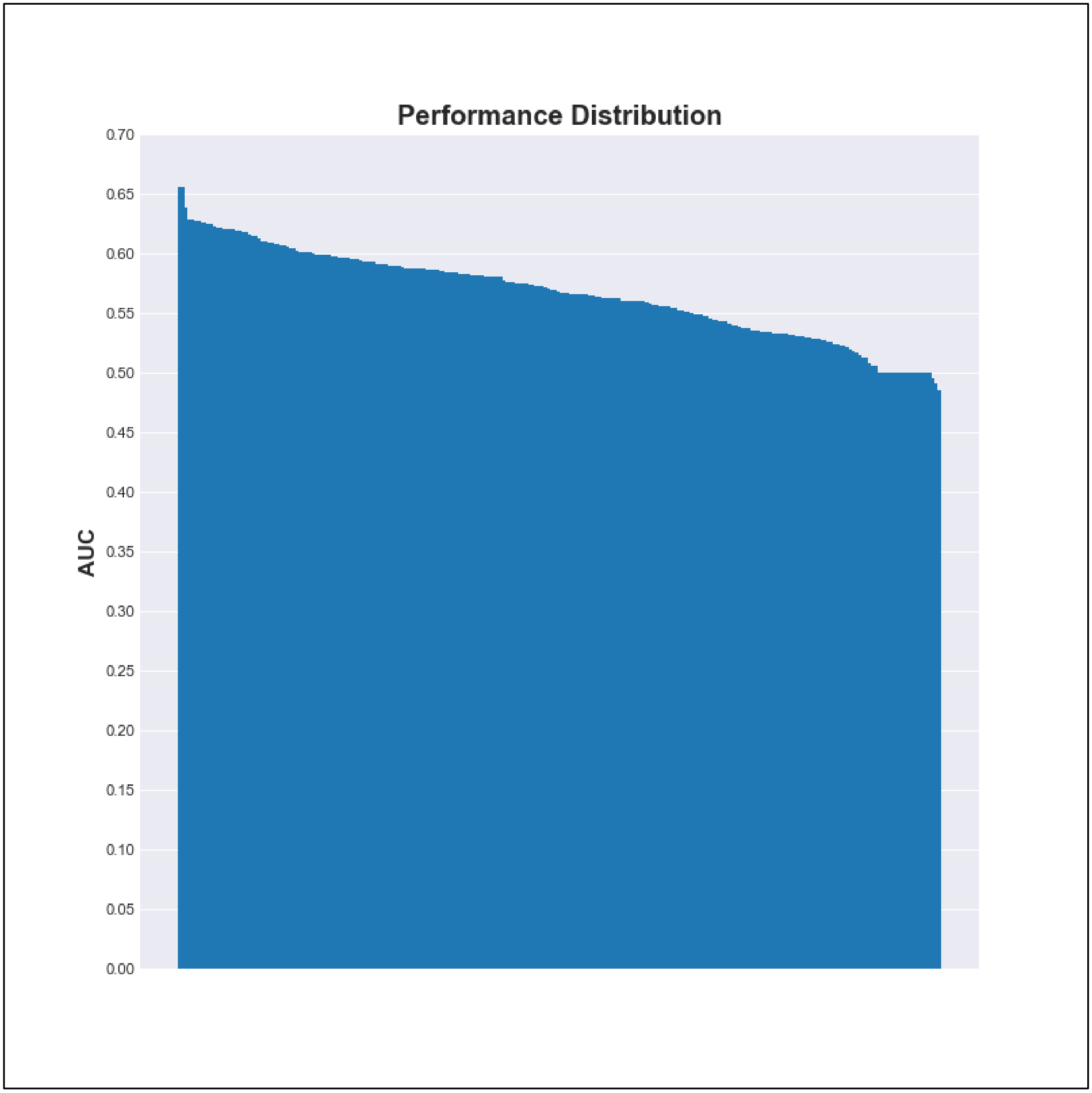
Each black bar represents a pipeline that include the component(s) of interest

Chi-Square + No Data Transform

PCA

The pipelines are sorted by performance with the high performing pipelines toward the left and the low performance pipelines toward the right

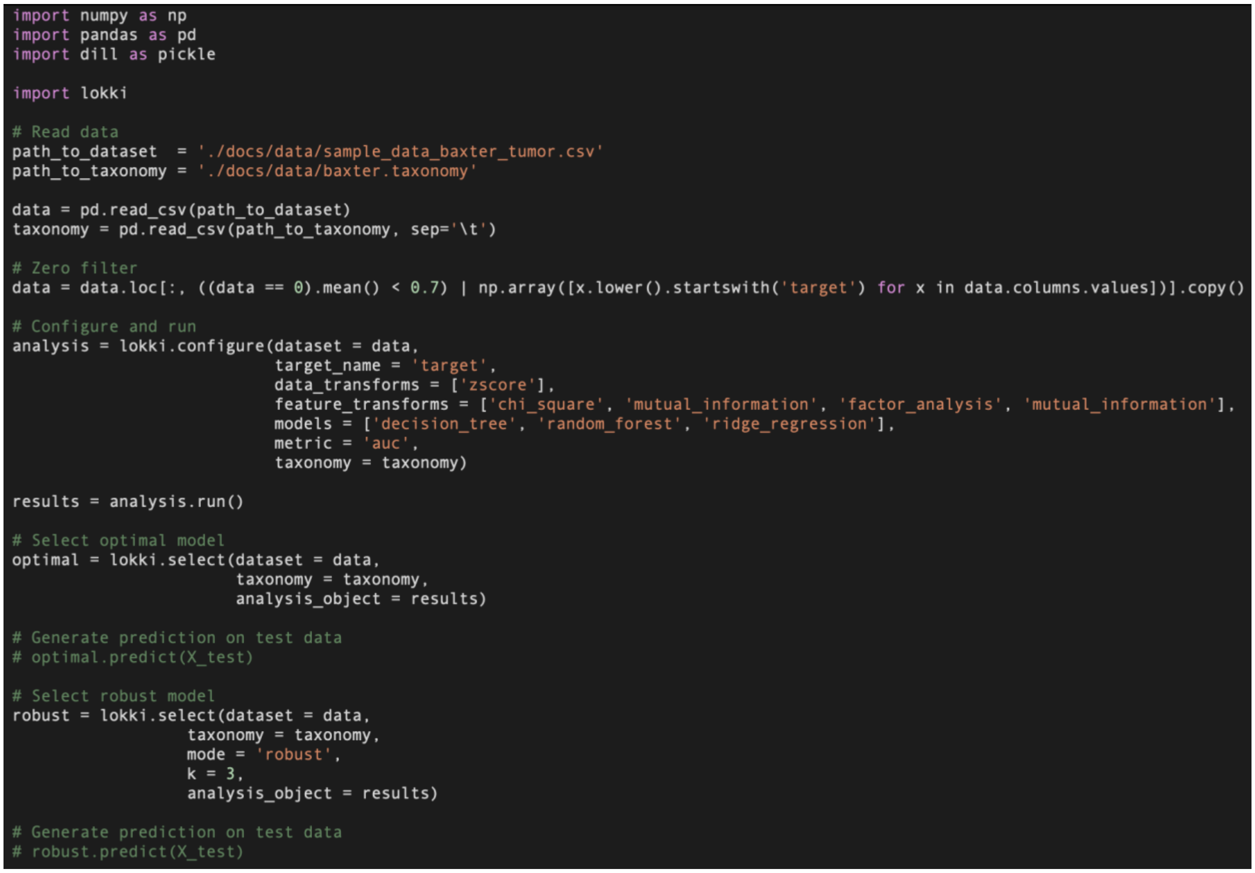
Figure 3c


Performance Distribution
AUC
0.70
0.65
0.60
0.55
0.50
0.45
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00

Model Selection

There are currently two methods of selecting a model for use on future dataset: Optimal and Robust. The **optimal** mode will select the pipeline with the best cross validated performance. The **robust** mode requires an additional parameter **k**, then the software will walk down the list of sorted list of pipelines and select the data transformation, feature selection or engineering, and model that first appears k times.

Figure 4

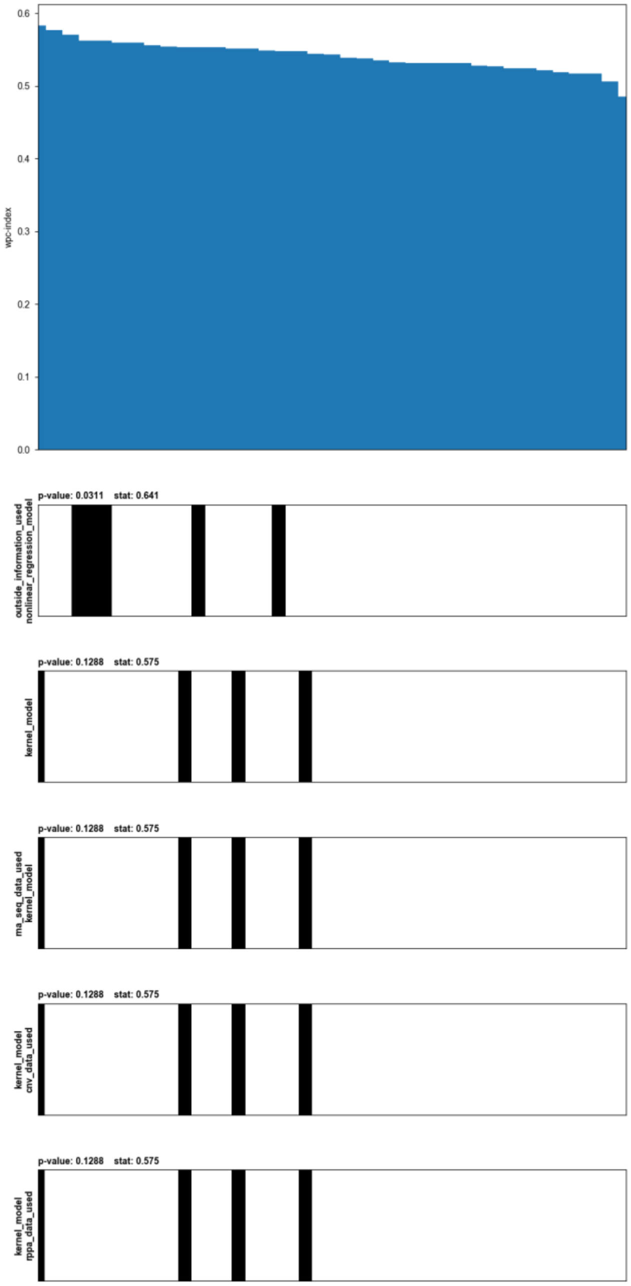


```python
import numpy as np
import pandas as pd
import dill as pickle

import lokki

# Read data
path_to_dataset  = './docs/data/sample_data_baxter_tumor.csv'
path_to_taxonomy = './docs/data/baxter.taxonomy'

data = pd.read_csv(path_to_dataset)
taxonomy = pd.read_csv(path_to_taxonomy, sep='\t')

# Zero filter
data = data.loc[:, ((data == 0).mean() < 0.7) | np.array([x.lower().startswith('target') for x in data.columns.values])].copy()

# Configure and run
analysis = lokki.configure(dataset = data,
                           target_name = 'target',
                           data_transforms = ['zscore'],
                           feature_transforms = ['chi_square', 'mutual_information', 'factor_analysis', 'mutual_information'],
                           models = ['decision_tree', 'random_forest', 'ridge_regression'],
                           metric = 'auc',
                           taxonomy = taxonomy)

results = analysis.run()

# Select optimal model
optimal = lokki.select(dataset = data,
                       taxonomy = taxonomy,
                       analysis_object = results)

# Generate prediction on test data
# optimal.predict(X_test)

# Select robust model
robust = lokki.select(dataset = data,
                  taxonomy = taxonomy,
                  mode = 'robust',
                  k = 3,
                  analysis_object = results)

# Generate prediction on test data
# robust.predict(X_test)
```

Analyzing Precomputed Performance

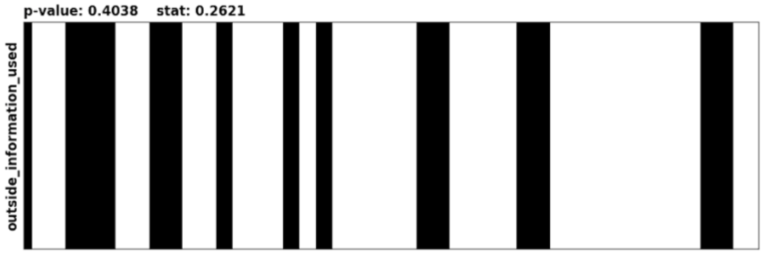
Users can also analyze precomputed performance files. For example, consider a recent DREAM challenge (1) where participants performed drug sensitivity prediction. It would be useful to know which characteristics appear across the different approaches. From the figure below, we can see that most of the most significant characteristic were 1) the use of outside information (in the form of annotated biological pathways) and 2) non-linear regression model.

Figure 5a



The individual enrichment plots for outside_information (Figure 5b) shows an in-significant result which suggests that in this study the quality of external biological pathway annotations are important, but only if combined with a non-linear model that can assimilate the information.

Figure 5b



References

Costello, J. C., Heiser, L. M., Georgii, E., Gönen, M., Menden, M. P., Wang, N. J., ... & Kallioniemi, O. (2014). A community effort to assess and improve drug sensitivity prediction algorithms. Nature biotechnology, 32(12), 1202-1212