

# The effect of pipeline-collection-diversity on performance

Sean Carver @ Data Machines Corporation

August 31, 2020

## Abstract

Do performers who submit *diverse* collections of pipelines tend to perform better than those who submit less diverse collections of pipelines? The answer for the Winter 2020 evaluation is effectively no. While there does exist statistically significant effect of increased diversity on increased score, the effect size is trivial. Thus, it seems that less diverse collections of pipelines, i.e. those that focus more on hyperparameter tuning, can as effective (or almost as effective, on average) as those that explore a greater diversity and/or ordering of primitives.

## 1 Introduction

In section 2, *Measures of Diversity*, we define several measures of the diversity within a collection of three or more pipelines. In section 3, *Results at a Glance*, we present a heat map showing briefly the (weak) connection between diversity and performance. In section 4, *Quantifying the Effect of Diversity*, we report on regression results which point to the measurable, though substantively trivial, effect of diversity on score. In a final section 5, *Visualizing Collections* we show a visualization of collections of pipelines for a problem together with a multiple alignment, for context.

## 2 Measures of Diversity

Before we can define *diversity* of a collection of (say 3 to 20) pipelines, we first need to define the distance between a pair of pipelines. We choose the Levenshtein edit distance as this measure between two pipelines. Specifically, we first express the pipelines as sequences of primitives, where primitives are written as “letters” in a large alphabet. The software we use accommodates all D3M primitives with two-letter pairs, each pair representing a single token (primitive) of the alphabet. The Levenshtein edit distance is the minimum number of substitutions, insertions and/or deletions needed to bring one sequence to coincide with the other. This measure satisfies the axioms for a distance.

But *distance* involves just two pipelines; *diversity* measures variation among a collection of three or more pipelines. We tried several alternatives for quantifying diversity. The most well-behaved measures (i.e. the ones that behave most closely to our expectations on synthetic data) were vector norms where the vector components were the Levenshtein distances for all possible unordered pairs of pipelines in the collection.

More precisely, we used  $l_p$  norms where  $p$  was a parameter varying between 1 and  $\infty$ . The different norms measure slightly different quantities. At the extreme, the  $l_\infty$  norm (maximum edit-distance component) is large when there is at least one pair of pipelines at great distance, regardless of the positions (great or small) of the other as-close or closer distances. On the other hand, the  $l_1$  norm (sum of the edit-distance components) can still be relatively large if there are many pairs pipelines at moderate distance, even though there is no pair at great distance. We point out that the choice the different norms can sometimes matter: we have noted that the choice can order sets of collections—synthetic or real—differently in terms of diversity.

In the  $l_p$  norm, what value for the parameter  $p$  do we pick? If you are using only one measure of diversity, we recommend the  $l_2$  norm as a nice trade off between the two extremes. That said, it is more informative to report two or more measures of diversity, in which case  $l_1$  and  $l_\infty$  should be preferred because they are most independent.

### 3 Results at a Glance

Figure 1 shows the diversity of performer pipeline submissions by problem type. The color-coding indicates diversity, with purple colors indicating small  $l_2$  edit distance norms, trending towards yellow colors indicating larger  $l_2$  edit distance norms. The number of asterisks in each cell captures the count of times a pipeline in that performer-category was the best (or tied). For instance, in the second row of the first column, UCB twice had the best-performing pipeline on a data set in the binary\_classification problem category. Overall, the figure suggests (and we quantify below) a very small but measurable effect of diversity.

### 4 Quantifying the Effect of Diversity

We use regression-based methods to quantify the effect of diversity on performance. We address complications by combining several overlapping methodologies. Overall, our goal is to specify predictive models of performance. Our unit of analysis is a collection of pipelines submitted by a performer for a particular problem. The response variable is the best score of all pipelines in the collection. Our models predict best score based on three explanatory variables: the  $l_1$  and  $l_\infty$  norms for the collection (as discussed above), as well as “count,” the number of pipelines in the collection.

The fact that different metrics were used for scoring different problems forced us to make adjustments. In some cases the best score was the greatest score, whereas in others the best score was the least score. By reversing the sign where appropriate, we transform scores so that greatest score was always best score. Our response variable was then called “max\_score.”

The fact that max\_scores were on widely different scales across problems forced us to switch to standardized coordinates (z-scores) for max\_score. The population for this standardization was all collections for all performers for one particular problem—as these were guaranteed to have metrics on the same scale.

At this point, we also transformed the predictor variables to standardized coordinates (against the same population), making predictors comparable only to other predictors for that problem.

Below, we report on five classes (A-E) of statistical models. For the purposes of analysis, we grouped problems by eleven categories (e.g. time series, binary classification, etc). See the horizontal axis of Figure 1 for a list of our categories.

In the first model, we group all problem categories together for analysis in a “complete pooling” model; see MODEL A.

In the second, we ran a fixed effects model (MODEL B) on category.

Third, we then ran a no pooling model (MODEL C), which is actually numerous separate models, one for each category.

We then ran a hierarchical model with random effects on the intercept (MODEL D).

Finally, we ran a hierarchical model with random effects on both the intercept and slope (MODEL E).

In all models, we see that there is a significant effect of  $l_1$  z-score diversity on performance, but the coefficient is small (less than .2). The fact that we are using z-scores for both the predictors and response variable complicates the interpretation of this result. What it says is that for an increase of z-score of the predictor variable ( $l_1$  z-score) by 1 the model predicts that the response variable (max\_score z-score) will increase by almost 0.2. Put another way when the raw  $l_1$  increases by one standard deviation for that problem, max\_score increases by almost 0.2 standard deviations for that problem.

It is helpful to translate this result into raw scores for  $l_1$ . A unit increase in  $l_1$  means an one extra substitution, insertion, or deletion is needed to bring just one pair of pipelines to coincidence. We keep z-scores for max\_score because raw scores for max\_scores are meaningless for across-problem comparison. To convert to a transformation from a unit increase in raw  $l_1$  to a specified z-score increase of max\_score, we must divide the coefficient (about 0.2, depending on model) by the standard deviation of  $l_1$  for the population referenced by the z-score. Of course, each problem has a different reference population so each problem has a different standard deviation. By implication, the transformed slopes will be different for each problem, but we can obtain summaries of the distribution of these slopes.

The 5-number summary of this distribution of slopes is

$$[0.00037622, 0.00052519, 0.00067828, 0.00114945, 0.00809511],$$

showing that a unit increase in  $l_1$  leads to at most a 8-thousandths increase in the z-score of max\_score, and the increase is usually quite a bit less. Our conclusion is that diversity has a negligibly small effect

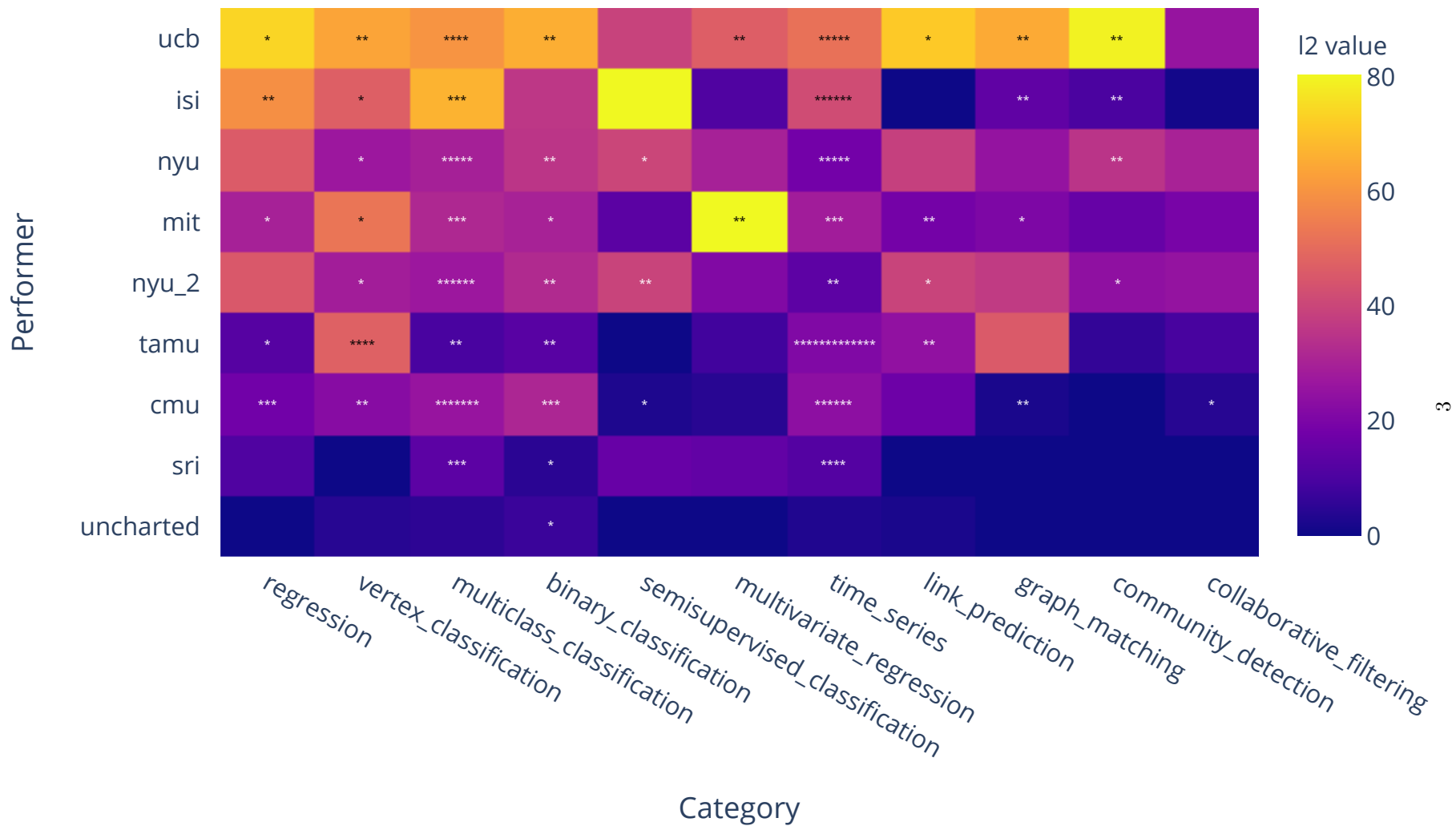


Figure 1: Heat map of  $l_2$  diversity see section “Measures of Diversity” for a discussion of this and other measures we use to quantify diversity. The horizontal axis is the problem category and the vertical axis is performer for the Winter 2020 evaluation. The number of asterisks in each cell is the number of problem instances in the corresponding category-performer which was the best (or tied for best) performing pipeline for any problem. There were 37 ties (including multi-way ties), corresponding to an additional 37 asterisks in the figure beyond the total number of 103 problems.

Model	Coefficient	p-value	estimate	std error
A	$l_1$ (z-score)	0.00716 **	1.984e-01	7.355e-02
B	$l_1$ (z-score)	0.00763 **	1.984e-01	7.414e-02
C	$l_1$ (z-score)	multiple models, same story.		
D	$l_1$ (z-score)	0.00716 **	1.984e-01	7.355e-02
E	$l_1$ (z-score)	0.0155 *	1.778e-01	7.329e-02

Table 1: Summary of models predicting the z-score of a performer’s best score in a submitted collection, based on  $l_1$  and  $l_\infty$  measures of diversity of the collection and count of number of pipelines in the collection. We performed a regression analysis with 5 different models (A-E, see text). The table shows the model, the coefficient examined (in this case, always considered just  $l_1$ , among other predictors we used:  $l_\infty$  and count). Furthermore, the table shows p-values (two tailed, under the null hypothesis that coefficient is zero), estimates of coefficients and standard errors of the estimates. Note that several entries of the table are identical to our numerical precision. We attribute this result to category-level predictors that do not covary with response variable. We examine the effect of colinearity between  $l_1$  and  $l_\infty$  and found that it did not change our results substantially. Significance codes: 0 “\*\*\*” 0.001 “\*\*” 0.01 “\*” 0.05 “.” 0.1 [blank] 1.

on performance.

Table 1 summarizes our results.

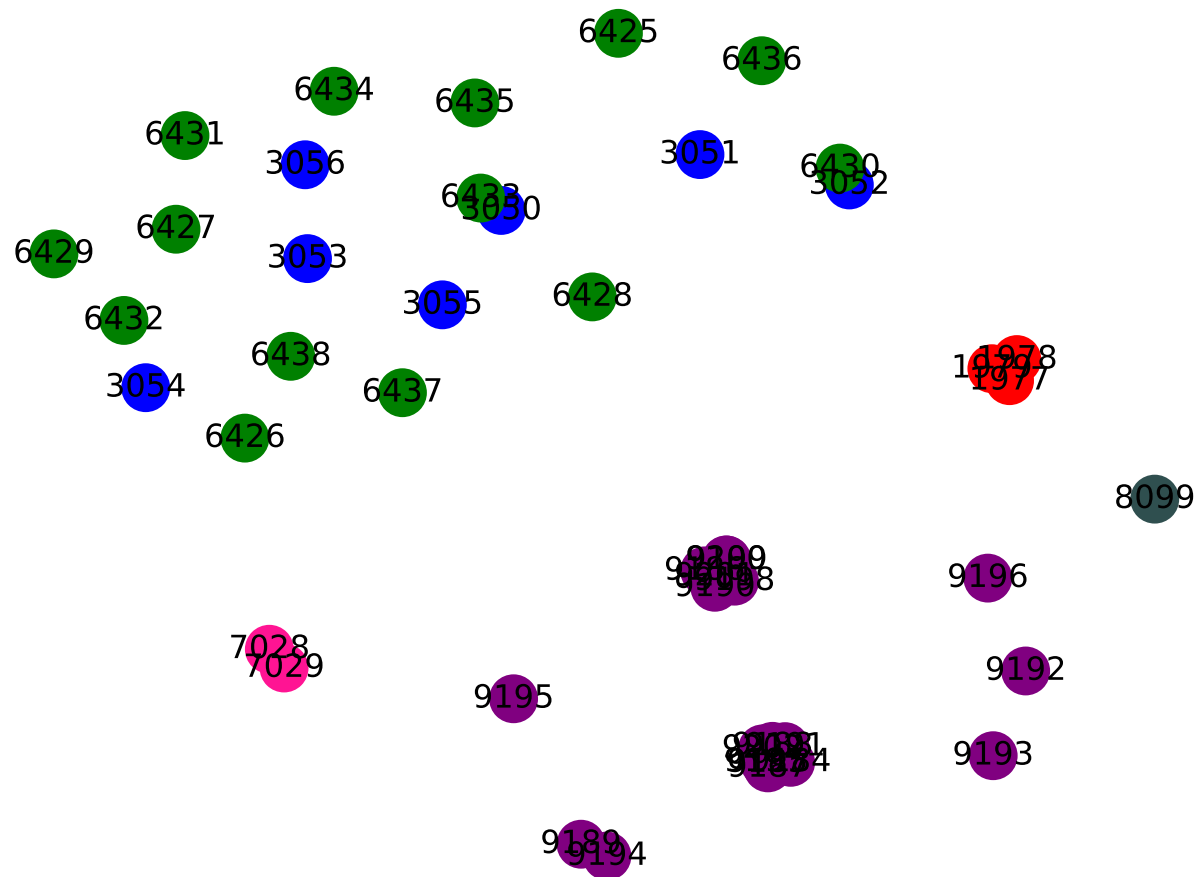
## 5 Visualizing Collections

In this final section, we show a cartoon visualization of collections of pipelines for a problem together with a multiple alignment (see Figure 2, and its caption).

# See Following Pages

Figure 2: The next several pages show a cartoon figure of a multiple sequence alignment of all submitted pipelines (all performers) for one randomly selected problem **semi\_1217\_click\_prediction\_small\_MIN\_METADATA**. The circles in the first pane illustrate the pipelines and are referenced with the number that corresponds to the label in the multiple alignment in the second pane. The colors correspond to different performers (also labeled in the second pane). The alphabet used to label primitives is shown on the third pane, which was randomly generated. The position of the circles was determined by a spring layout procedure with pipelines close in edit distance supposedly remain close as nodes. That said, usually impossible solve this problem perfectly (at least in two dimensions) and we have noted that the software returns some misleading results while presumably trying to draw a pretty picture (what it was designed to do). Nevertheless, we found these figures useful for keeping pipelines in memory—as long as the image is interpreted as a “cartoon” and not a “topographical map.” Cartoons for the other problems are available upon request.

SEMI\_1217\_click\_prediction\_small\_MIN\_METADATA



cmu	pipeline_1977	CX-SN-----IB----ZK-XD-----ZK-DD-LW-EX-GT-MM
cmu	pipeline_1978	CX-SN-----IB----ZK-XD-----ZK-DD-LW-EX-GT-MM
cmu	pipeline_1979	CX-SN-----IB----ZK-XD-----ZK-DD-LW-EX-GT-MM
nyu_2	pipeline_3050	CX-SN-HH-HH-HH-HH-HH-HH-XD-ZK-ZK----ZK-LW-JP-VV----ZK-VK-GT-MM-----
nyu_2	pipeline_3051	CX-SN-HH-HH-HH-HH-HH-HH-XD-ZK-DD-ZK-ZK-LW-JP-EX----ZK-UA-MM-----
nyu_2	pipeline_3052	CX-SN-HH-HH-HH-HH-HH-HH-XD-ZK-DD-ZK-ZK-LW-JP-EX----ZK-UO-MM-----
nyu_2	pipeline_3053	CX-SN-HH-HH-HH-HH-HH-HH-XD-ZK-ZK----ZK-LW-JP-VV-EX-ZK-VK-GT-MM-----
nyu_2	pipeline_3054	CX-SN-HH-HH-HH-HH-HH-HH-XD-ZK-ZK----ZK-LW-JP-VV-NC-ZK-VK-GT-MM-----
nyu_2	pipeline_3055	CX-SN-HH-HH-HH-HH-HH-HH-XD-ZK-ZK----ZK-LW-JP-DD----ZK-VK-GT-MM-----
nyu_2	pipeline_3056	CX-SN-HH-HH-HH-HH-HH-HH-XD-ZK-ZK----ZK-LW-JP-VV-XN-ZK-VK-GT-MM-----
nyu	pipeline_6425	CX-SN-HH-HH-HH-HH-HH-HH-XD-ZK-DD-ZK-LW-ZK-JP-EX----ZK-UA-MM-----
nyu	pipeline_6426	CX-SN-HH-HH-HH-HH-HH-HH-XD-ZK-ZK----ZK-LW-JP-VV-EX-ZK-OC-GT-MM-----
nyu	pipeline_6427	CX-SN-HH-HH-HH-HH-HH-HH-XD-ZK-ZK----ZK-LW-JP-VV-NC-ZK-OD-GT-MM-----
nyu	pipeline_6428	CX-SN-HH-HH-HH-HH-HH-HH-XD-ZK-ZK----ZK-LW-JP-VV----ZK-OC-GT-MM-----
nyu	pipeline_6429	CX-SN-HH-HH-HH-HH-HH-HH-XD-ZK-ZK----LW-ZK-JP-DD----ZK-VK-GT-MM-----
nyu	pipeline_6430	CX-SN-HH-HH-HH-HH-HH-HH-XD-ZK-DD-ZK-ZK-LW-JP-EX----ZK-UO-MM-----
nyu	pipeline_6431	CX-SN-HH-HH-HH-HH-HH-HH-XD-ZK-ZK----ZK-LW-JP-DD-EX-ZK-YX-GT-MM-----
nyu	pipeline_6432	CX-SN-HH-HH-HH-HH-HH-HH-XD-ZK-ZK----ZK-LW-JP-DD-EX-ZK-OD-GT-MM-----
nyu	pipeline_6433	CX-SN-HH-HH-HH-HH-HH-HH-XD-ZK-ZK----ZK-LW-JP-VV----ZK-VK-GT-MM-----

## SEMI\_1217\_click\_prediction\_small\_MIN\_METADATA

nyu	pipeline_6434	CX-SN-HH-HH-HH-HH-HH-HH-XD-ZK-ZK----	ZK-LW-JP-DD-EX-ZK-VK-GT-MM-----
nyu	pipeline_6435	CX-SN-HH-HH-HH-HH-HH-HH-XD-ZK-ZK----	ZK-LW-JP-VV-----ZK-OD-GT-MM-----
nyu	pipeline_6436	CX-SN-HH-HH-HH-HH-HH-HH-XD-ZK-ZK-LW-ZK-ZK-JP-DD-----	UM-GT-MM-----
nyu	pipeline_6437	CX-SN-HH-HH-HH-HH-HH-HH-XD-ZK-ZK----	ZK-LW-JP-DD-----ZK-YC-GT-MM-----
nyu	pipeline_6438	CX-SN-HH-HH-HH-HH-HH-HH-XD-ZK-ZK----	ZK-LW-JP-DD-----ZK-TX-GT-MM-----
uncharted	pipeline_7028	CX-SN-----	BD-XD-DD-RT-----ZK-ZK-XH-MM-----
uncharted	pipeline_7029	CX-SN-----	BD-XD-DD-RT-----ZK-ZK-XH-MM-----
tamu	pipeline_8099	---SN-----	IB-XD-ZK-ZK-----DD-JO-GT-MM-----
sri	pipeline_9184	NI-SN-----	IB-XD-ZK-KG-----ZK-GT-UG-WE-MM---
sri	pipeline_9185	NI-SN-----	IB-XD-ZK-KG-----ZK-GT-UG-WE-MM---
sri	pipeline_9186	NI-SN-----	IB-XD-ZK-KG-----ZK-GT-UG-IP-MM---
sri	pipeline_9187	NI-SN-----	IB-XD-ZK-KG-----ZK-GT-UG-WE-MM---
sri	pipeline_9188	NI-SN-----	IB-XD-ZK-KG-----ZK-GT-UG-WE-MM---
sri	pipeline_9189	NI-SN-----	IB-XD-ZK-KG-----ZK-GT-UG-LW-JA-MM
sri	pipeline_9190	NI-SN-----	IB-XD-ZK-KG-----ZK-GT-UG-IP-MM---
sri	pipeline_9191	NI-SN-----	IB-XD-ZK-KG-----ZK-GT-UG-WE-MM---
sri	pipeline_9192	NI-SN-----	IB-XD-ZK-KG-----ZK-GT-UG-XJ-WE-MM
sri	pipeline_9193	NI-SN-----	IB-XD-ZK-KG-----ZK-GT-UG-LW-WE-MM
sri	pipeline_9194	NI-SN-----	IB-XD-ZK-KG-----ZK-GT-UG-LW-JA-MM
sri	pipeline_9195	NI-SN-----	IB-XD-ZK-KG-----ZK-GT-UG-AG-MM---
sri	pipeline_9196	NI-SN-----	IB-XD-ZK-KG-----ZK-GT-UG-OD-IP-MM
sri	pipeline_9197	NI-SN-----	IB-XD-ZK-KG-----ZK-GT-UG-WE-MM---
sri	pipeline_9198	NI-SN-----	IB-XD-ZK-KG-----ZK-GT-UG-IP-MM---
sri	pipeline_9199	NI-SN-----	IB-XD-ZK-KG-----ZK-GT-UG-IP-MM---
sri	pipeline_9200	NI-SN-----	IB-XD-ZK-KG-----ZK-GT-UG-IP-MM---
sri	pipeline_9201	NI-SN-----	IB-XD-ZK-KG-----ZK-GT-UG-IP-MM---
sri	pipeline_9202	NI-SN-----	IB-XD-ZK-KG-----ZK-GT-UG-WE-MM---

AB: Multi Table Deep Feature Synthesis  
AG: sklearn.ensemble.bagging.BaggingClassifier  
AO: OWLRegression  
AQ: jhu.gclust  
AW: sklearn.decomposition.fastica\_.FastICA  
AX: sklearn.preprocessing.data.StandardScaler  
BB: CorexContinuous  
BD: simon  
BH: sklearn.neural\_network.multilayer\_perceptron.MLPClassifier  
BL: Columns text reader  
BN: SDNE  
BO: Annotated tabular extractor  
BP: DeepAR  
BU: Gaussian Latent Dirichlet Allocation Topic Modelling  
CI: sklearn.linear\_model.passive\_aggressive.PassiveAggressiveClassifier  
CK: sklearn.impute.SimpleImputer  
CL: sklearn.svm.classes.SVR  
CM: DSBox Unary Data Encoder  
CR: sklearn.tree.tree.DecisionTreeClassifier  
CS: sklearn.preprocessing.\_encoders.OrdinalEncoder  
CT: find projections  
CV: sklearn.ensemble.forest.RandomForestRegressor  
CX: Denormalize datasets  
DA: gator  
DD: sklearn.impute.SimpleImputer  
DE: sklearn.kernel\_ridge.KernelRidge  
DG: DSBox Image Featurizer VGG16  
DI: sklearn.linear\_model.base.LinearRegression  
DL: STMBplus\_auto feature selector  
DN: jhu.link\_pred\_rc  
DP: kanine  
DR: Random forest classifier  
DT: Random Sampling Imputer  
DX: sklearn.linear\_model.coordinate\_descent.ElasticNet  
DZ: Dataset Metafeature Extraction  
EA: tsne  
EH: Randomized Principal Component Analysis using Polynomial Features  
EO: XGBoost GBTree classifier  
EP: Binary encoder  
ES: Extracts columns by structural type  
EW: lupi\_svm.LupiSvmClassifier  
EX: sklearn.preprocessing.data.RobustScaler  
EY: K means  
FD: DSBox ensemble voting  
FM: Label encoder with an unseen category  
FN: DSBox Profiler  
FP: ISI DSBox To Numeric DataFrame  
FT: LightGBM GBTree classifier  
FU: sklearn.svm.classes.LinearSVR  
FX: sklearn.kernel\_approximation.RBFSampler  
GD: sklearn.svm.classes.SVC  
GP: DSBox horizontal concat  
GS: DataFrame to ndarray converter  
GT: Iterative labeling for semi-supervised learning  
HA: DSBox feature scaler  
HC: jhu.ase  
HG: ndarray to Dataframe converter  
HH: Add semantic types to columns  
HJ: sklearn.linear\_model.bayes.ARDRegression  
HL: Tensor Machine Regularized Least Squares  
HM: Removes columns  
HN: jhu.link\_pred\_graph\_reader  
HQ: DSBox vertically concat  
HR: lupi\_mfa.lupi\_mfa.LupiMFA  
HW: sklearn.random\_projection.SparseRandomProjection



IA: sklearn.ensemble.forest.RandomTreesEmbedding  
IB: Determine missing semantic types for columns automatically  
IH: Load single graph and dataframe into a parseable object  
IM: DSBox Splitter  
IP: sklearn.ensemble.gradient\_boosting.GradientBoostingClassifier  
IQ: sklearn.dummy.DummyClassifier  
IR: sklearn.discriminant\_analysis.LinearDiscriminantAnalysis  
IS: ISI DSBox Data Encoder  
IT: DSBox random projection timeseries featurization  
JA: sklearn.neighbors.nearest\_centroid.NearestCentroid  
JH: CommunityDetection  
JK: Enrich dates  
JM: sklearn.gaussian\_process.gpr.GaussianProcessRegressor  
JN: EchoLinearRegression  
JO: Text encoder  
JP: Concatenate multiple dataframes  
JW: sklearn.linear\_model.stochastic\_gradient.SGDRegressor  
KA: sklearn.dummy.DummyRegressor  
KF: Gaussian Mixture Models  
KG: Autoflow Data Conditioner  
KL: sklearn.discriminant\_analysis.QuadraticDiscriminantAnalysis  
KR: sklearn.ensemble.forest.ExtraTreesRegressor  
KS: retina\_net  
KV: Map DataFrame resources to new resources using provided primitive  
KW: sklearn.preprocessing.data.MaxAbsScaler  
LB: XGBoost GBTree regressor  
LC: Replace semantic types for columns  
LF: Hierarchical Dirichlet Process Topic Modelling  
LI: Latent Dirichlet Allocation Topic Modelling  
LL: sklearn.linear\_model.ridge.Ridge  
LM: Grouping Field Compose  
LT: Text Classifier  
LW: sklearn.preprocessing.data.OneHotEncoder  
LY: DataFrame Sampling  
MF: SeededGraphMatcher  
ML: Data cleaning  
MM: Construct pipeline predictions output  
MP: K-means Clustering  
MU: sklearn.decomposition.truncated\_svd.TruncatedSVD  
MV: sklearn.random\_projection.GaussianRandomProjection  
MY: Nearest Neighbor Classification with Cover Trees  
NA: find projections  
NC: Regex dataset filter  
ND: Nearest Neighbor Regressor with Cover Trees  
NF: Label decoder for UnseenLabelEncoderPrimitive  
NG: DSBox Iterative Regression Imputer  
NI: Columns text reader  
NK: General Relational Dataset  
NP: Image Transfer  
NT: sklearn.preprocessing.data.MinMaxScaler  
NU: RFM Preconditioned Polynomial Kernel Ridge Regression  
NY: sklearn.naive\_bayes.GaussianNB  
OB: sklearn.linear\_model.coordinate\_descent.LassoCV  
OC: sklearn.feature\_selection.variance\_threshold.VarianceThreshold  
OD: sklearn.feature\_selection.univariate\_selection.SelectPercentile  
OO: DSBox Arima Primitive  
OP: CorexText  
OQ: sklearn.ensemble.weight\_boosting.AdaBoostClassifier  
PA: Columns image reader  
PC: sklearn.feature\_selection.univariate\_selection.GenericUnivariateSelect  
PE: jhu.gclass  
PK: sklearn.ensemble.forest.ExtraTreesClassifier  
PN: sklearn.ensemble.weight\_boosting.AdaBoostRegressor  
PS: Huber PCA  
PY: sklearn.linear\_model.least\_angle.Lars

QM: sklearn.impute.MissingIndicator  
QN: Preprocessing for categorical columns  
QO: Extracts columns  
QQ: DSBox Timeseries Featurizer dataframe to List Transformer  
QT: RF Features  
QU: RFM Preconditioned Gaussian Kernel Ridge Regression  
RC: sklearn.linear\_model.stochastic\_gradient.SGDClassifier  
RE: VAR  
RF: sklearn.tree.tree.DecisionTreeRegressor  
RL: lupi\_rf.LupiRFCClassifier  
RN: Load audio collection from dataset into a single dataframe  
RQ: sklearn.naive\_bayes.MultinomialNB  
RS: Single Table Deep Feature Synthesis  
RT: hdbscan  
SB: sklearn.neighbors.regression.KNeighborsRegressor  
SC: sklearn.ensemble.bagging.BaggingRegressor  
SD: DSBox Object Detection YOLO  
SK: List encoder  
SN: Extract a DataFrame from a Dataset  
SP: Robust Sparse Principal Component Analysis  
SR: Time series formatter  
ST: sklearn.feature\_extraction.text.CountVectorizer  
SU: Sparse Principal Component Analysis  
SW: Select dataframe from list of dataframes  
SY: DSBox Image Featurizer ResNet50  
TA: BERT pair classification  
TD: Audio Transfer  
TE: lstm\_fcn  
TH: sklearn.cluster.hierarchical.FeatureAgglomeration  
TK: DSBox Cleaning Featurizer  
TL: find projections numeric  
TN: Dataset sampling primitive  
TV: DSBox Greedy Imputer  
TW: sklearn.decomposition.kernel\_pca.KernelPCA  
TX: JMIplus\_auto feature selector  
UA: sklearn.ensemble.forest.RandomForestClassifier  
UD: Remove semantic types from columns  
UG: Parses strings into their types  
UI: sent2vec\_wrapper  
UJ: Tree-Augmented Naive Bayes Classifier  
UL: find projections  
UM: sklearn.feature\_selection.univariate\_selection.SelectFwe  
UN: DSBox do-nothing primitive  
UO: sklearn.neighbors.classification.KNeighborsClassifier  
UR: sklearn.decomposition.pca.PCA  
UV: sklearn.linear\_model.coordinate\_descent.Lasso  
UX: Categorical imputer  
UY: Replace singeltons  
UZ: VertexNomination  
VB: LinkPrediction  
VH: Random Classifier  
VK: PCA Features  
VO: tsne  
VP: DSBox Image Featurizer dataframe to tensor transformer  
VS: Casts DataFrame  
VT: Load graphs into a parseable object  
VV: DSBox feature labeler  
VW: Collaborative filtering  
WA: jhu.sgm  
WB: Matrix Completion via Sparse Factorization  
WC: sklearn.neural\_network.multilayer\_perceptron.MLPRegressor  
WE: sklearn.naive\_bayes.BernoulliNB  
WI: Echo  
WJ: sklearn.preprocessing.data.QuantileTransformer  
WK: Extract a list of Graphs from a Dataset

WR: sklearn.feature\_extraction.text.TfidfVectorizer  
WU: jhu.lcc  
WY: sklearn.linear\_model.passive\_aggressive.PassiveAggressiveRegressor  
WZ: Concatenate two dataframes  
XB: lupi\_rfsel.LupiRFSelClassifier  
XD: Parses strings into their types  
XH: EnsembleForest  
XI: Low Rank Imputer  
XJ: sklearn.preprocessing.data.PolynomialFeatures  
XK: sklearn.kernel\_approximation.Nystroem  
XN: sklearn.preprocessing.data.Binarizer  
XS: sklearn.ensemble.gradient\_boosting.GradientBoostingRegressor  
XT: One-hot encoder  
XV: Load edgelist into a parseable object  
XZ: sklearn.linear\_model.logistic.LogisticRegression  
YB: sklearn.preprocessing.data.Normalizer  
YC: S2TMBplus feature selector  
YJ: sklearn.svm.classes.LinearSVC  
YM: One-hot maker  
YN: jhu.lse  
YS: DSBox do-nothing primitive dataset version  
YX: Feature Selection  
ZB: XGBoost DART classifier  
ZF: DSBox Mean Imputer  
ZK: Extracts columns by semantic type  
ZR: Pandas one hot encoder  
ZZ: Perform dataset augmentation using Datamart