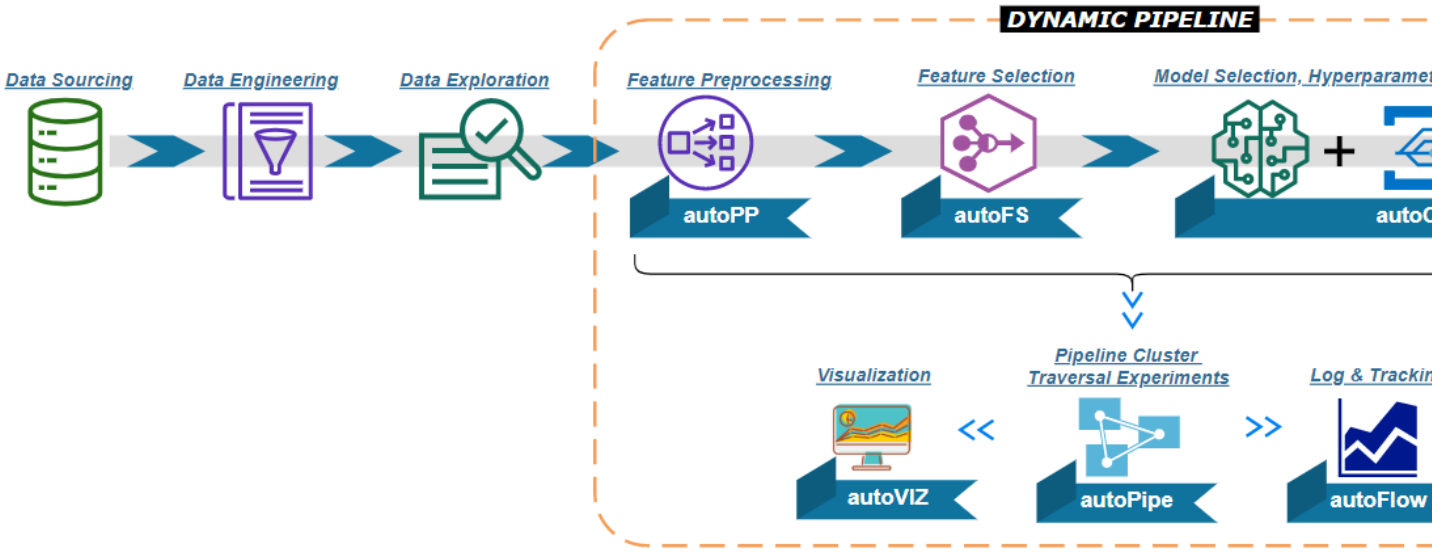
Dynamic Pipeline

Aug 10, 2020

Contents

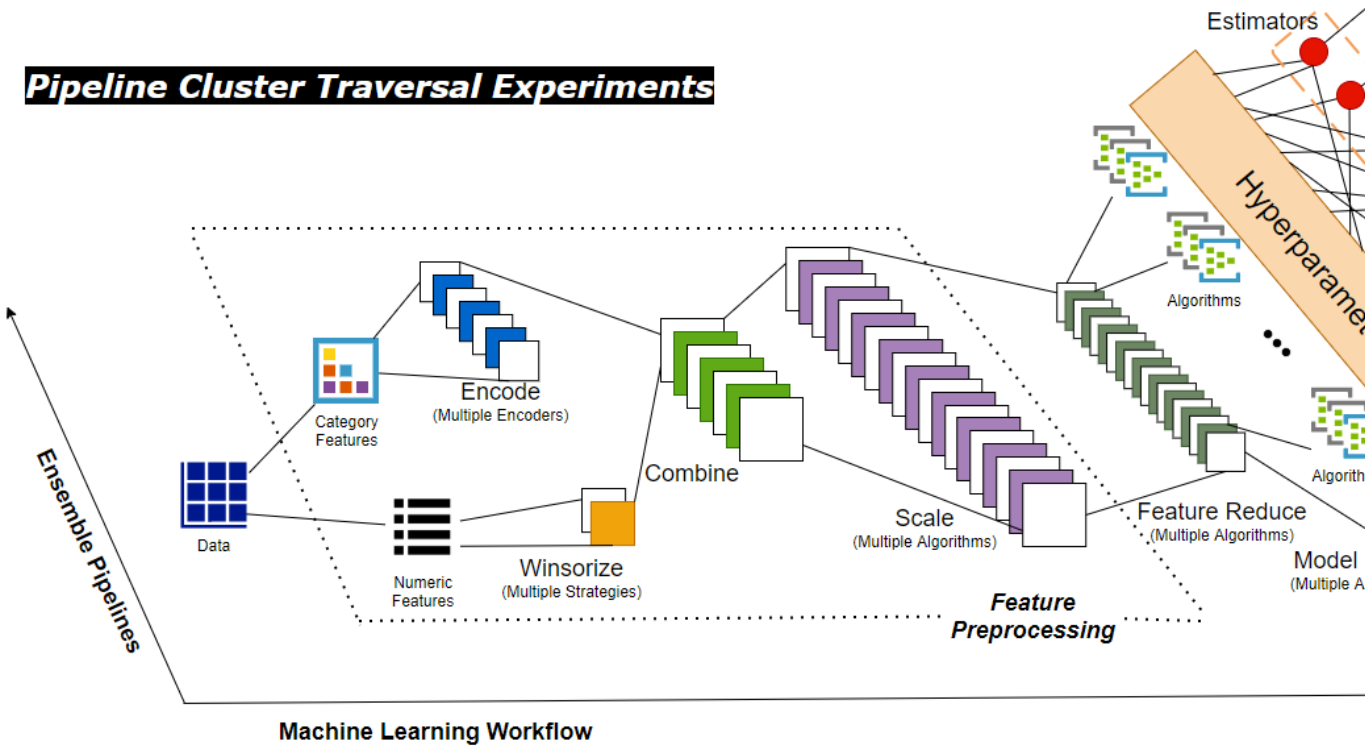
Dynamic Pipeline is a high-level API toolkit to help data scientists building models in ensemble way, and automating Machine Learning workflow with simple codes. Comparing other popular "AutoML or Automatic Machine Learning" APIs, **Dynamic Pipeline** is designed as an omni-ensembled ML workflow optimizer with higher-level API targeting to avoid manual repetitive train-along-evaluate experiments in general pipeline building.

To achieve that, **Dynamic Pipeline** creates *Pipeline Cluster Traversal Experiments* to assemble all cross-matching pipelines covering major tasks of Machine Learning workflow, and apply traversal-experiment to search the optimal baseline model. Besides, by modularizing all key pipeline components in reusable packages, it allows all components to be custom tunable along with high scalability.



The core concept in **Dynamic Pipeline** is *Pipeline Cluster Traversal Experiments*, which is a theory, first raised by Tony Dong during Genpact 2020 GVector Conference, to optimize and automate Machine Learning Workflow using ensemble pipelines algorithm.

Comparing other automatic or classic machine learning workflow's repetitive experiments using single pipeline, *Pipeline Cluster Traversal Experiments* is more powerful, with larger coverage scope, to find the best model without manual intervention, and also more flexible with elasticity to cope with unseen data due to its ensemble designs in each component.



In summary, **Dynamic Pipeline** shares a few useful properties for data scientists:

- *Easy & less coding* - High-level APIs to implement *Pipeline Cluster Traversal Experiments*, and each ML component is highly automated and modularized;
- *Well ensembled* - Each key component is ensemble of popular algorithms w/ optimal hyperparameters tuning included;
- *Hardly omission* - *Pipeline Cluster Traversal Experiments* are designed to cross-experiment with combined permuted input datasets, feature selection, and model selection;
- *Scalable & Consistency* - Each module could add new algorithms easily due to its ensemble & reusable design; no extra needs to modify existing codes for new experiment
- *Adaptable* - *Pipeline Cluster Traversal Experiments* makes it easier to adapt unseen datasets with the right pipeline;
- *Custom Modify Welcomed* - Support custom settings to add/remove algorithms or modify hyperparameters for elastic requirements.

CHAPTER 1

Installation

To install Dynamic Pipeline, run this command in your terminal:

```
$ pip install dynapipe
```

This is the preferred method to install Dynamic Pipeline, as it will always install the most recent stable release.

CHAPTER 2

autoPP Module

Description :

- **This module is used for data preprocessing operation:**
  - Impute with missing value
  - Winsorize with outlier
  - Scaling using popular scaler approaches
  - Encoding category features using popular encoder approaches
  - Generated all combination datasets for further modeling and evaluation
  - Sparsity calculation as the critera for output datasets filtering
  - Custom parameters initial settings, add/remove winsorization, scaling, or encoding strategies.
- **Class:**
  - **dynaPreprocessing** [Focus on classification/regression prprocessing problems]
    - fit() - fit & transform method for preprocessing
- **Current available strategies:**
  - **Scaling** [Numeric features scaling, default settings]
    - "standard" : StandardScaler() approach
    - "minmax" - MinMaxScaler() approach
    - "maxabs" - MaxAbsScaler() approach
    - "robust" - RobustScaler() approach
  - **Encoding** [Category features encoding, default settings]
    - "onehot" : OnehotEncoder() approach, with dummy trap consideration in regression problem
    - "label" : LabelEncoder() approach

* "frequency" : Frequency calculation approach
* "mean" : Mean calculation approach

– **winsorization** [Default limits settings]

* (0.01,0.01) : Top 1% and bottom 1% will be excluded
* (0.05,0.05) : Top 5% and bottom 5% will be excluded

2.1 dynapipePreprocessing

class dynapipe.autoPP.**dynaPreprocessing**(*custom_parameters=None*, *label_col=None*, *model_type='reg'*, *export_output_files=False*)

Automated feature preprocessing including imputation, winsorization, encoding, and scaling in ensemble algorithms, to generate permutation input datasets for further pipeline components.

Parameters

- **custom_parameters** (*dictionary, default = None*) – Custom parameters settings input.

  NOTE: default_parameters = { "scaler" : ["None", "standard", "minmax", "maxabs", "robust"], "encode_band" : [10], "low_encode" : ["onehot","label"], "high_encode" : ["frequency", "mean"], "winsorizer" : [(0.01,0.01),(0.05,0.05)], "sparsity" : [0.40], "cols" : [30] }

- **label_col** (*str, default = None*) – Name of label column.
- **model_type** (*str, default = "reg"*) – "reg" for regression problem or "cls" for classification problem - Default: "reg".
- **export_output_files** (*bool, default = False*) – Export qualified permutated datasets to ./df_folder.

Example

References

None

fit(*input_data=None*)

Fits and transforms a pandas dataframe to non-missing values, outlier excluded, categories encoded and scaled datasets by all algorithms permutation.

Parameters **input_data** (*pandas dataframe, shape = [n_samples, n_features]*) – NOTE: The input_data should be the datasets after basic data cleaning & well feature deduction, the more features involve will result in more columns permutation outputs.

Returns

- **DICT_PREP_DF** (*dictionary*) – Each key is the # of output preprocessed dataset, each value stores the dataset
- **DICT_PREP_INFO** (*dictionary*) – Dictionary for reference. Each key is the # of the output preprocessed dataset, each value stores the column names of the dataset
- *NOTE - Log records will generate and save to ./logs folder automatedly.*

2.2 PPtools

class dynapipe.funcPP.**PPtools**(*data=None*, *label_col=None*, *model_type='reg'*)

This class stores feature preprocessing transform tools.

Parameters

- **data** (*df, default = None*) – Pre-cleaned dataset for feature preprocessing.
- **label_col** (*str, default = None*) – Name of label column.
- **model_type** (*str, default = "reg"*) – Value in ["reg","cls"]. The "reg" for regression problem, and "cls" for classification problem.

Example

References

None

encode_tool(*en_type=None*, *category_col=None*)

Category features encoding, included: "onehot" - OneHot algorithm; "label" - LabelEncoder algorithm; "frequency" - Frequency Encoding algorithm; "mean" - Mean Encoding algorithm.

Parameters **en_type** (*str, default = None*) – Value in ["reg","cls"]. Will drop first encoded column to cope with dummy trap issue, when value is "reg".

Returns

Return type Encoded column/dataset for each category feature

impute_tool()

Imputation with the missing values.

Parameters **None** –

Returns

Return type None

remove_feature(*feature_name*)

Remove feature.

Parameters **feature_name** (*str/list, default = None*) – column name, or list of column names wants to extract.

Returns

Return type None

remove_zero_col_tool(*data=None*)

Remove the columns with all value zero.

Parameters **data** (*pandas dataset, default = None*) – dataset needs to remove all zero columns

Returns

Return type All-zero-columns dataset

scale_tool(*df=None*, *sc_type=None*)

Feature scaling.

Parameters

- **df** (*df, default = None*) – Dataset wants to be scaled
- **sc_type** (*str, default = None*) – Value in ["None","standard","minmax","maxabs","robust"]. Select which scaling algorithm: "None" - No scale algorithm apply; "standard" - StandardScaler algorithm; "minmax" - MinMaxScaler algorithm; "maxabs" - MaxAbsScaler algorithm; "RobustScaler" - RobustScaler algorithm

Returns

Return type Scaled dataset

sparsity_tool(*data=None*)

Calculate the sparsity of the datset.

Parameters **data** (*df, default = None*) –

Returns

Return type Value of sparsity

split_category_cols()

Split input datasets to numeric dataset and category dataset.

Parameters **None** –

Returns

Return type None

winsorize_tool(*lower_ban=None*, *upper_ban=None*)

Feature outliers excluding with winsorization.

Parameters

- **lower_ban** (*float, default = None*) – Bottom percent of excluding data needs to set here.
- **upper_ban** – Top percent of excluding data needs to set here.

Returns

Return type None

CHAPTER 3

autoFS Module

Description :

- **This module is used for features selection:**
  - Automate the feature selection with several selectors
  - Evaluate the outputs from all selector methods, and ranked a final list of the top important features
- **Class:**
  - **dynaFS_clf** [Focus on classification problems]
    - fit() - fit and transform method for classifier
  - **dynaFS_reg** [Focus on regression problems]
    - fit() - fit and transform method for regressor
- **Current available selectors**
  - **clf_fs** [Class focusing on classification features selection]
    - kbest_f : SelectKBest() with f_classif core
    - kbest_chi2 - SelectKBest() with chi2 core
    - rfe_lr - RFE with LogisticRegression() estimator
    - rfe_svm - RFE with SVC() estimator
    - rfecv_svm - RFECV with SVC() estimator
    - rfe_tree - RFE with DecisionTreeClassifier() estimator
    - rfecv_tree - RFECV with DecisionTreeClassifier() estimator
    - rfe_rf - RFE with RandomForestClassifier() estimator
    - rfecv_rf - RFECV with RandomForestClassifier() estimator
  - **reg_fs** [Class focusing on regression features selection]
    - kbest_f : SelectKBest() with f_regression core

* rfe_svm : RFE with SVC() estimator
* rfecv_svm : RFECV with SVC() estimator
* rfe_tree : RFE with DecisionTreeRegressor() estimator
* rfecv_tree : RFECV with DecisionTreeRegressor() estimator
* rfe_rf : RFE with RandomForestRegressor() estimator
* rfecv_rf : RFECV with RandomForestRegressor() estimator

3.1 dynaFS_clf

class dynapipe.autoFS.**dynaFS_clf**(*fs_num=None*, *random_state=None*, *cv=None*, *in_pipeline=False*, *input_from_file=True*)

This class implements feature selection for classification problem.

Parameters

- **fs_num** (*int, default = None*) – Set the # of features want to select out.
- **random_state** (*int, default = None*) – Random state value.
- **cv** (*int, default = None*) – # of folds for cross-validation.
- **in_pipeline** (*bool, default = False*) – Should be set to "True" when using autoPipe module to build Pipeline Cluster Traveral Experiments.
- **input_from_file** (*bool, default = True*) – When input dataset is dataframe, needs to set "True"; Otherwise, i.e. array, needs to set "False".

Example

References

None

fit(*tr_features*, *tr_labels*)

Fits and transforms a dataframe with built-in algorithms, to select top features.

Parameters

- **tr_features** (*df, default = None*) – Train features columns. (NOTE: In the Pipeline Cluster Traversal Experiments, the features columns should be from the same pipeline dataset).
- **tr_labels** (*array/df, default = None*) – Train label column, when input_from_file = True, must be pandas datframe. (NOTE: In the Pipeline Cluster Traversal Experiments, the label column should be from the same pipeline dataset).

Returns

- **fs_num** (*int*) – # of top features has been select out.
- **fs_results** (*array*) – Selected & ranked top feature names.
- *NOTE - Log records will generate and save to ./logs folder automatedly.*

3.2 dynaFS_reg

class dynapipe.autoFS.**dynaFS_reg**(*fs_num=None*, *random_state=None*, *cv=None*, *in_pipeline=False*, *input_from_file=True*)

This class implements feature selection for regression problem.

Parameters

- **fs_num** (*int, default = None*) – Set the # of features want to select out.
- **random_state** (*int, default = None*) – Random state value.
- **cv** (*int, default = None*) – # of folds for cross-validation.
- **in_pipeline** (*bool, default = False*) – Should be set to "True" when using autoPipe module to build Pipeline Cluster Traveral Experiments.
- **input_from_file** (*bool, default = True*) – When input dataset is dataframe, needs to set "True"; Otherwise, i.e. array, needs to set "False".

Example

References

None

fit(*tr_features*, *tr_labels*)

Fits and transforms a dataframe with built-in algorithms, to select top features.

Parameters

- **tr_features** (*df, default = None*) – Train features columns. (NOTE: In the Pipeline Cluster Traversal Experiments, the features columns should be from the same pipeline dataset).
- **tr_labels** (*array/df, default = None*) – Train label column, when input_from_file = True, must be pandas datframe. (NOTE: In the Pipeline Cluster Traversal Experiments, the label column should be from the same pipeline dataset).

Returns

- **fs_num** (*int*) – # of top features has been select out.
- **fs_results** (*array*) – Selected & ranked top feature names.
- *NOTE - Log records will generate and save to ./logs folder automatedly.*

3.3 clf_fs

class dynapipe.selectorFS.**clf_fs**(*fs_num=None*, *random_state=None*, *cv=None*)

This class stores classification selectors.

Parameters

- **fs_num** (*int, default = None*) – Set the # of features want to select out.
- **random_state** (*int, default = None*) – Random state value.
- **cv** (*int, default = None*) – # of folds for cross-validation.

Example

References

None

3.4 reg_fs

class dynapipe.selectorFS.**reg_fs**(*fs_num*, *random_state=None*, *cv=None*)
This class stores regression selectors.

Parameters

- **fs_num** (*int, default = None*) – Set the # of features want to select out.
- **random_state** (*int, default = None*) – Random state value.
- **cv** (*int, default = None*) – # of folds for cross-validation.

Example

References

None

CHAPTER 4

autoCV Module

Description :

- **This module is used for model selection:**
  - Automate the models training with cross validation
  - GridSearch the best parameters
  - Export the optimized models as pkl files, and saved them in /pkl folders
  - Validate the optimized models, and select the best model
- **Class**
  - **dynaClassifier** [Focus on classification problems]
    - fit() : fit method for classifier
  - **dynaRegressor** [Focus on regression problems]
    - fit() : fit method for regressor
- **Current available estimators**
  - **clf_cv** [Class focusing on classification estimators]
    - lgr - Logistic Regression (aka logit, MaxEnt) classifier - LogisticRegression()
    - svm : C-Support Vector Classification - SVM.SVC()
    - mlp : Multi-layer Perceptron classifier - MLPClassifier()
    - ada : An AdaBoost classifier - AdaBoostClassifier()
    - rf : Random Forest classifier - RandomForestClassifier()
    - gb : Gradient Boost classifier - GradientBoostingClassifier()
    - xgb : XGBoost classifier - xgb.XGBClassifier()
  - **reg_cv** [Class focusing on regression estimators]
    - lr : Linear Regression - LinearRegression()

* knn : Regression based on k-nearest neighbors - KNeighborsRegressor()
* svr : Epsilon-Support Vector Regression - SVM.SVR()
* rf : Random Forest Regression - RandomForestRegressor()
* ada : An AdaBoost regressor - AdaBoostRegressor()
* gb : Gradient Boosting for regression -GradientBoostingRegressor()
* tree : A decision tree regressor - DecisionTreeRegressor()
* mlp : Multi-layer Perceptron regressor - MLPRegressor()
* xgb : XGBoost regression - XGBRegressor()
* hgboost : Hist Gradient Boosting regression - HistGradientBoostingRegressor(); New added on 8/7/2020
* huber : Huber regression - HuberRegressor(); New added on 8/7/2020
* rgcv : Ridge cross validation regression - RidgeCV(); New added on 8/7/2020
* cvlasso : Lasso cross validation regression - LassoCV(); New added on 8/7/2020
* sgd : Stochastic Gradient Descent regression - SGDRegressor(); New added on 8/7/2020

4.1 dynaClassifier

class dynapipe.autoCV.**dynaClassifier**(*custom_estimators=None*, *random_state=13*, *cv_num=5*, *in_pipeline=False*, *input_from_file=True*)

This class implements classification model selection with hyperparameters grid search and cross-validation.

Parameters

- **custom_estimators** (*list, default = None*) – Custom set the estimators in the autoCV regression module(if set None, will use all available estimators). Current version's default available estimators are ['lgr','svm','mlp','rf','ada','gb','xgb'].
- **random_state** (*int, default = None*) – Random state value.
- **cv** (*int, default = None*) – # of folds for cross-validation.
- **in_pipeline** (*bool, default = False*) – Should be set to "True" when using autoPipe module to build Pipeline Cluster Traveral Experiments.
- **input_from_file** (*bool, default = True*) – When input dataset is df, needs to set "True"; Otherwise, i.e. array, needs to set "False".

Example

References

None

fit(*tr_features=None*, *tr_labels=None*)

Fit and train datasets with classification hyperparameters GridSearch and CV across multiple estimators. Module will Auto save trained model as {estimator_name}_clf_model.pkl file to ./pkl folder. :param features: Train features columns. (NOTE: In the Pipeline Cluster Traversal Experiments, the features columns should be from the same pipeline dataset). :type features: df, default = None :param labels: Train label column.

(NOTE: In the Pipeline Cluster Traversal Experiments, the label column should be from the same pipeline dataset).

Returns

- **cv_num** (*int*) – # of fold for cross-validation.
- **DICT_EST** (*dictionary*) – key is the name of estimators, value is the ralated trained model
- *NOTE - Trained model auto save function only avalable when in_pipeline = "False".*
- *NOTE - Log records will generate and save to ./logs folder automatedly.*

4.2 dynaRegressor

class dynapipe.autoCV.**dynaRegressor**(*custom_estimators=None*, *random_state=25*, *cv_num=5*, *in_pipeline=False*, *input_from_file=True*)

This class implements regression model selection with with hyperparameters grid search and cross-validation. Module will Auto save trained model as {estimator_name}_reg_model.pkl file to ./pkl folder.

Parameters

- **custom_estimators** (*list, default = None*) – Custom set the estimators in the autoCV regression module(if set None, will use all available estimators). Current version's default available estimators are ['lr','knn','tree','svm','mlp','rf','gb','ada','xgb','hgboost','huber','rgcv','cvlasso','sgd'].
- **random_state** (*int, default = None*) – Random state value.
- **cv** (*int, default = None*) – # of folds for cross-validation.
- **in_pipeline** (*bool, default = False*) – Should be set to "True" when using autoPipe module to build Pipeline Cluster Traveral Experiments.
- **input_from_file** (*bool, default = True*) – When input dataset is df, needs to set "True"; Otherwise, i.e. array, needs to set "False".

Example

References

None

fit(*tr_features=None*, *tr_labels=None*)

Fit and train datasets with regression hyperparameters GridSearch and CV across multiple estimators.

Parameters

- **features** (*df, default = None*) – Train features columns. (NOTE: In the Pipeline Cluster Traversal Experiments, the features columns should be from the same pipeline dataset).
- **labels** (*df ,default = None*) – Train label column. (NOTE: In the Pipeline Cluster Traversal Experiments, the label column should be from the same pipeline dataset).

Returns

- **cv_num** (*int*) – # of fold for cross-validation.
- **DICT_EST** (*dictionary*) – key is the name of estimators, value is the ralated trained model.

- *NOTE - Trained model auto save function only avalable when in_pipeline = "False".*
- *NOTE - Log records will generate and save to ./logs folder automatedly.*

4.3 evaluate_model

class dynapipe.autoCV.**evaluate_model**(*model_type=None*, *in_pipeline=False*)
This class implements model evaluation and return key score results.

Parameters

- **model_type** (*str, default = None*) – Value in ["reg","cls"]. The "reg" for regression problem, and "cls" for classification problem.
- **in_pipeline** (*bool, default = False*) – Should be set to "True" when using autoPipe module to build Pipeline Cluster Traveral Experiments.

Example

References

fit(*name=None*, *model=None*, *features=None*, *labels=None*)
Model evaluation with all models by the validate datasets.

Parameters

- **name** (*str, default = None*) – Estimator name.
- **model** (*pkl, default = None*) – Model needs to evaluate. Needs pkl file as input when in_pipeline = "False"; otherwise, should use DICT_EST[estimator name] as the input here.
- **features** (*df, default = None*) – Validate features columns. (NOTE: In the Pipeline Cluster Traversal Experiments, the features columns should be from the same pipeline dataset).
- **labels** (*df ,default = None*) – Validate label column. (NOTE: In the Pipeline Cluster Traversal Experiments, the label column should be from the same pipeline dataset).

Returns optimal_scores – When model_type = "cls", will return [name,accuracy,precision,recall,latency] info of model validation results. when model_type = "reg", will return [name,R-squared,MAE,MSE,RMSE,latency] info of model validation results.

Return type list

4.4 clf_cv

class dynapipe.estimatorCV.**clf_cv**(*cv_val=None*, *random_state=None*)
This class stores classification estimators.

Parameters

- **random_state** (*int, default = None*) – Random state value.
- **cv_val** (*int, default = None*) – # of folds for cross-validation.

Example

References

None

4.5 reg_cv

class dynapipe.estimatorCV.**reg_cv**(*cv_val=None*, *random_state=None*)
This class stores regression estimators.

Parameters

- **random_state** (*int, default = None*) – Random state value.
- **cv_val** (*int, default = None*) – # of folds for cross-validation.

Example

References

None

4.6 data_splitting_tool

dynapipe.utilis_func.**data_splitting_tool**(*feature_cols=None*, *label_col=None*, *val_size=0.2*, *test_size=0.2*, *random_state=13*)
Splitting each pipeline's dataset into train, validate, and test parts for Pipeline Cluster Traversal Experiments.

NOTE: When in_pipeline = "True", this function will be built-in function in autoPipe module. So it needs to use pipeline_splitting_rule() to setup splitting rule.

Parameters

- **label_col** (*array/df, default = None*) – Column of label.
- **feature_cols** (*df, default = None*) – Feature columns.
- **val_size** (*float, default = 0.2*) – Value within [0~1]. Percentage of validate data. NOTE - When val_size with no input value will not return X_val & y_val
- **test_size** (*float, default = 0.2*) – Value within [0~1]. Percentage of test data.
- **random_state** (*int, default = 13*) – Random state value.

Returns

- **X_train** (*array*) – Train features dataset
- **y_train** (*array*) – Train label dataset
- **X_val** (*array*) – Validate features datset
- **y_val** (*array*) – Validate label dataset
- **X_test** (*array*) – Test features dataset
- **y_test** (*array*) – Test label dataset

4.7 reset_parameters

dynapipe.utilis_func.**reset_parameters**()
Reset autoCV estimators hyperparameters and searching range to default values.

Parameters **None** –

Returns

Return type None

Example

4.8 update_parameters

dynapipe.utilis_func.**update_parameters**(*mode='None'*, *estimator_name='None'*, ***kwargs*)
Update autoCV estimators hyperparameters and searching range to custom values.

NOTE: One line of command could only update one estimator.

Parameters

- **mode** (*str, default = None*) – Value in ["cls","reg"]. "cls" will modify classification estimators; "reg" will modify regression estimators.
- **estimator_name** (*str, default = None*) – Name of estimator.
- ****kwargs** (*list, default = None*) – Lists of values using comma splitting, i.e. C=[0.1,0.2],kernel=["linear"].

Returns

Return type None

Example

4.9 export_parameters

dynapipe.utilis_func.**export_parameters**()
Export current autoCV estimators hyperparameters and searching range to current work dictionary.

Parameters **None** –

Returns

Return type None

Example

4.10 Defaults Parameters for Classifiers/Regressors

Estimators default parameters setting:

Table 1: Classifiers Estimators Default Parameters Searching Range

Estimators:	Parameters:	Value Range:
lgr	'C'	[0.001, 0.01, 0.1, 1, 10, 100, 1000]
svm	'C'	[0.1, 1, 10]
	'kernel'	['linear', 'poly', 'rbf', 'sigmoid']
mlp	'activation'	['identity','relu', 'tanh', 'logistic']
	'hidden_layer_sizes'	[10, 50, 100]
	'learning_rate'	['constant', 'invscaling', 'adaptive']
	'solver'	['lbfgs', 'sgd', 'adam']
ada	'n_estimators'	[50,100,150]
	'learning_rate'	[0.01,0.1, 1, 5, 10]
rf	'max_depth'	[2, 4, 8, 16, 32]
	'n_estimators'	[5, 50, 250]
gb	'n_estimators'	[50,100,150,200,250,300]
	'max_depth'	[1, 3, 5, 7, 9]
	'learning_rate'	[0.01, 0.1, 1, 10, 100]
xgb	'n_estimators'	[50,100,150,200,250,300]
	'max_depth'	[3, 5, 7, 9]
	'learning_rate'	[0.01, 0.1, 0.2,0.3,0.4]

Table 2: Regressors Default Parameters Searching Range

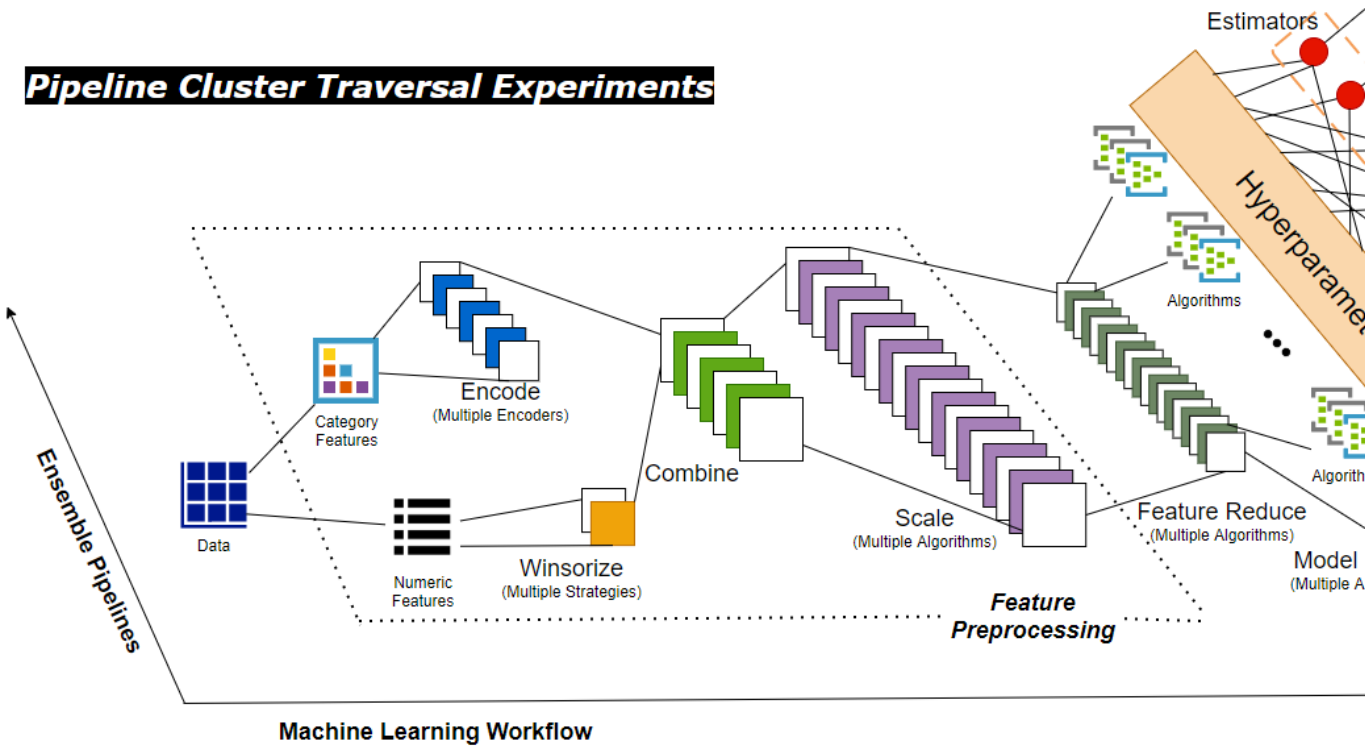
Estimators:	Parameters:	Value Range:
lr	'normalize'	[True,False]
svm	'C'	[0.1, 1, 10]
	'kernel'	['linear', 'poly', 'rbf', 'sigmoid']
mlp	'activation'	['identity','relu', 'tanh', 'logistic']
	'hidden_layer_sizes'	[10, 50, 100]
	'learning_rate'	['constant', 'invscaling', 'adaptive']
	'solver'	['lbfgs', 'adam']
ada	'n_estimators'	[50,100,150,200,250,300]
	'loss'	['linear','square','exponential']
	'learning_rate'	[0.01, 0.1, 0.2,0.3,0.4]
tree	'splitter'	['best', 'random']
	'max_depth'	[1, 3, 5, 7, 9]
	'min_samples_leaf'	[1,3,5]
rf	'max_depth'	[2, 4, 8, 16, 32]
	'n_estimators'	[5, 50, 250]
gb	'n_estimators'	[50,100,150,200,250,300]
	'max_depth'	[3, 5, 7, 9]
	'learning_rate'	[0.01, 0.1, 0.2,0.3,0.4]
xgb	'n_estimators'	[50,100,150,200,250,300]
	'max_depth'	[3, 5, 7, 9]
	'learning_rate'	[0.01, 0.1, 0.2,0.3,0.4]
sgd	'shuffle'	[True,False]
	'penalty'	['l2', 'l1', 'elasticnet']
	'learning_rate'	['constant','optimal','invscaling']
cvlasso	'fit_intercept'	[True,False]
rgcv	'fit_intercept'	[True,False]
huber	'fit_intercept'	[True,False]
hgboost	'max_depth'	[3, 5, 7, 9]
	'learning_rate'	[0.1, 0.2,0.3,0.4]

CHAPTER 5

autoPipe Module

Description :

- **This module is used to build *Pipeline Cluster Traversal Experiments*:**
  - Create sequential components of *Pipeline Cluster Traversal Experiments*
  - Apply traversal experiments through pipeline cluster to find the best baseline model
  - Generate comparable and parameter-tracable dictionaies and reports to support autoVIZ and autoFlow modules
- **Build Steps:**
  - autoPP - dynaPreprocessing() Class in autoPP module
  - Datasets Splitting - pipeline_splitting_rule() Function in utilis_funs module
  - autoFS - dynaFS_clf() or dynaFS_reg() Class in autoFS module
  - autoCV - dynaClassifier() or dynaRegressor() Class in autoCV module
  - Model Evaluate - evaluate_model() Class in autoCV module



5.1 autoPipe

class dynapipe.autoPipe.**autoPipe**(*steps*)

This class is to build Pipeline Cluster Traversal Experiments.

Parameters **steps** (*list, default = None*) – List of (name, transform) tuples (implementing fit & transform) that are chained, in the order in which they are chained, with the last object a model evaluation function.

Example

References

None

fit(*data*)

Fits and transforms a chain of Dynamic Pipeline modules.

Parameters **input_data** (*pandas dataframe, shape = [n_samples, n_features]*) – NOTE: The input_data should be the datasets after basic data cleaning & well feature deduction, the more features involve will result in more columns permutation outputs.

Returns

- **DICT_PREP_INFO** (*dictionary*) – Each key is the # of preprocessed dataset("Dataset_xxx" format, i.e. "Dataset_10"), each value stores an info string about what transforms applied. i.e. DICT_PREPROCESSING['Dataset_0'] stores value

"winsor_0-Scaler_None– Encoded Features:['diagnosis', 'Size_3', 'area_mean']", which means applied 1st mode of winsorization, none scaler applied, and the encoded columns names(shown the enconding approachs in the names)

- **DICT_FEATURE_SELECTION_INFO** (*dictionary*) – Each key is the # of preprocessed dataset, each value stores the name of features selected after the autoFS module.
- **DICT_MODELS_EVALUATION** (*dictionary*) – Each key is the # of preprocessed dataset, each value stores the model evaluation results with its validate dataset.
- **DICT_DATA** (*dictionary*) – Each key is the # of preprocessed dataset, and first level sub-key is the type of splitted sets(including 'DICT_Train','DICT_TEST',and'DICT_Validate'). The second level sub-key is "X" for features and "y" for label, each value stores the datasets related to the keys(Pandas Dataframe format) i.e. DICT_DATA['Dataset_0']['DICT_TEST']["X"] is the train features of Dataset_0's test dataset
- **models_summary** (*Pandas Dataframe*) – Model selection results ranking table among all composits of preprocessed datasets, selected features and all posible models with optimal parameters.
- *NOTE - Log records will generate and save to ./logs folder automatedly.*

CHAPTER 6

autoViz Module

Description :

- **This module is used for outputs visualization:**
  - Visualize and retrieve each pipeline's generating steps & performance;
  - Using this module as the open interface for further interactive visualization development.

6.1 autoViz

class dynapipe.autoViz.**autoViz**(*preprocess_dict=None*, *report=None*)

This class implements model visualization.

Parameters

- **preprocess_dict** (*dict, default = None*) – 1st output result (DICT_PREPROCESS) of autoPipe module.
- **report** (*df, default = None*) – 4th output result (dyna_report) of autoPipe module.

Example

References

clf_model_retrieval(*metrics=None*)

This function implements classification model retrieval visualization.

Parameters **metrics** (*str, default = None*) – Value in ["accuracy","precision","recall"].

Example

References

CHAPTER 7

autoFlow Module

Description :

- **This module is used for logging & model tracking:**
  - Log and retrieve each module's operation history & intermediate results;
  - Using this module as the open interface for further interactive model tracking development.

Here're samples of each module's log file:

- autoCV module: https://raw.githubusercontent.com/tonyleidong/DynamicPipeline/master/docs/autoCV_log_2020.08.07.17.28.34.log
- autoFS module: https://raw.githubusercontent.com/tonyleidong/DynamicPipeline/master/docs/autoFS_log_2020.07.16.12.25.48.log
- autoPP module: https://raw.githubusercontent.com/tonyleidong/DynamicPipeline/master/docs/autoPP_log_2020.08.07.17.28.34.log
- autoPipe module: https://raw.githubusercontent.com/tonyleidong/DynamicPipeline/master/docs/autoPipe_log_2020.08.07.17.28.34.log

CHAPTER 8

Examples

8.1 Feature preprocessing for a regression problem using autoPP:

Demo Code:

```
import pandas as pd
from funcPP import PPtools
from autoPP import dynaPreprocessing

df = pd.read_csv('../data/preprocessing/breast_cancer.csv')

custom_parameters = {
    "scaler" : ["None", "standard"],
     # threshold number of category dimension
    "encode_band" : [6],
     # low dimension encoding
    "low_encode" : ["onehot","label"],
     # high dimension encoding
    "high_encode" : ["frequency", "mean"],
    "winsorizer" : [(0.1,0.1)],
    "sparsity" : [0.4862],
    "cols" : [27]
   }
dyna = dynaPreprocessing(custom_parameters = custom_parameters, label_col = 'diagnosis
↪', model_type = "reg")
dict_df = dyna.fit(input_data = df)
print(f"Total combinations: {len(dict_df.keys())}")
print(dict_df['winsor_0-Scaler_standard-Dataset_441'])
```

Output:

```
 Now in Progress - Data Preprocessing Ensemble Iteration: Estimate about 0
Total combinations: 64
     diagnosis    Size_3  area_mean  compactness_mean  concave points_mea
```

(continues on next page)

(continued from previous page)

```
0            1  1.290564   1.151477          1.730765            1.63873
1            1 -1.423416   1.823311         -0.679975            0.53514
2            1 -0.066426   1.823311          1.163585            1.63873
3            1  1.290564  -1.011406          1.730765            1.63873
4            1 -0.066426   1.823311          0.548763            1.61037
..         ...       ...        ...               ...                ..
281          0 -0.066426  -0.866487         -1.300016           -0.80503
282          1 -0.066426   1.657991          0.807396            1.30604
283          1 -0.066426   0.462408          1.624135            1.17625
284          0 -0.066426  -0.552378         -0.290663           -0.60750
285          0  1.290564  -0.649460         -1.300016           -1.25679
[286 rows x 26 columns]
```

8.2 Features selection for a regression problem using autoFS:

Demo Code:

```
import pandas as pd
from dynapipe.autoFS import dynaFS_reg

tr_features = pd.read_csv('./data/regression/train_features.csv')
tr_labels = pd.read_csv('./data/regression/train_labels.csv')

# Set input_form_file = False, when label values are array. Select 'True' from Pandas␣
↪dataframe.
reg_fs_demo = dynaFS_reg( fs_num = 5,random_state = 13,cv = 5,input_from_file = True)
# Select detail_info = True, when you want to see the detail of the iteration
reg_fs_demo.fit(tr_features,tr_labels,detail_info = False)
```

Output:

```
*DynaPipe* autoFS Module ===> Selector kbest_f gets outputs: ['INDUS', 'NOX', 'RM',
↪'PTRATIO', 'LSTAT']
Progress: [###-----------------] 14.3%

*DynaPipe* autoFS Module ===> Selector rfe_svm gets outputs: ['CHAS', 'NOX', 'RM',
↪'PTRATIO', 'LSTAT']
Progress: [######--------------] 28.6%

*DynaPipe* autoFS Module ===> Selector rfe_tree gets outputs: ['CRIM', 'RM', 'DIS',
↪'TAX', 'LSTAT']
Progress: [#########-----------] 42.9%

*DynaPipe* autoFS Module ===> Selector rfe_rf gets outputs: ['CRIM', 'RM', 'DIS',
↪'PTRATIO', 'LSTAT']
Progress: [###########---------] 57.1%

*DynaPipe* autoFS Module ===> Selector rfecv_svm gets outputs: ['CRIM', 'ZN', 'INDUS',
↪ 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT']
Progress: [##############------] 71.4%

*DynaPipe* autoFS Module ===> Selector rfecv_tree gets outputs: ['CRIM', 'CHAS', 'NOX
↪', 'RM', 'AGE', 'DIS', 'TAX', 'PTRATIO', 'B', 'LSTAT']
Progress: [#################---] 85.7%
```

(continues on next page)

(continued from previous page)

```
*DynaPipe* autoFS Module ===> Selector rfecv_rf gets outputs: ['CRIM', 'ZN', 'NOX',
↪'RM', 'AGE', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT']
Progress: [####################] 100.0%

The DynaPipe autoFS identify the top 5 important features for regression are: ['RM',
↪'LSTAT', 'PTRATIO', 'NOX', 'CRIM'].
```

8.3 Model selection for a classification problem using autoCV:

Demo Code:

```
import pandas as pd
from dynapipe.autoCV import dynaClassifier,evaluate_clf_model
import joblib

tr_features = pd.read_csv('./data/classification/train_features.csv')
tr_labels = pd.read_csv('./data/classification/train_labels.csv')
val_features = pd.read_csv('./data/classification/val_features.csv')
val_labels = pd.read_csv('./data/classification/val_labels.csv')

# Set input_form_file = False, when label values are array. Select 'True' from Pandas␣
↪dataframe.
clf_cv_demo = dynaClassifier(random_state = 13,cv_num = 5,input_from_file = True)
# Select detail_info = True, when you want to see the detail of the iteration
clf_cv_demo.fit(tr_features,tr_labels,detail_info = False)

models = {}
for mdl in ['lgr','svm','mlp','rf','ada','gb','xgb']:
    models[mdl] = joblib.load('./pkl/{}_clf_model.pkl'.format(mdl))

for name, mdl in models.items():
    evaluate_clf_model(name, mdl, val_features, val_labels)
```

Output:

```
*DynaPipe* autoCV Module ===> lgr_CrossValidation with 5 folds:

Best Parameters: {'C': 1, 'random_state': 13}

Best CV Score: 0.7997178628107917

Progress: [###-----------------] 14.3%

*DynaPipe* autoCV Module ===> svm_CrossValidation with 5 folds:

Best Parameters: {'C': 0.1, 'kernel': 'linear'}

Best CV Score: 0.7959619114794568

Progress: [######--------------] 28.6%

*DynaPipe* autoCV Module ===> mlp_CrossValidation with 5 folds:
```

(continues on next page)

(continued from previous page)

```
Best Parameters: {'activation': 'tanh', 'hidden_layer_sizes': (50,), 'learning_rate':
↪'constant', 'random_state': 13, 'solver': 'lbfgs'}

Best CV Score: 0.8184094515958386

Progress: [#########----------] 42.9%

*DynaPipe* autoCV Module ===> rf_CrossValidation with 5 folds:

Best Parameters: {'max_depth': 4, 'n_estimators': 250, 'random_state': 13}

Best CV Score: 0.8240521953800035

Progress: [###########---------] 57.1%

*DynaPipe* autoCV Module ===> ada_CrossValidation with 5 folds:

Best Parameters: {'learning_rate': 0.1, 'n_estimators': 100, 'random_state': 13}

Best CV Score: 0.824034561805678

Progress: [##############------] 71.4%

*DynaPipe* autoCV Module ===> gb_CrossValidation with 5 folds:

Best Parameters: {'learning_rate': 0.01, 'max_depth': 3, 'n_estimators': 300, 'random_
↪state': 13}

Best CV Score: 0.8408746252865456

Progress: [#################---] 85.7%

*DynaPipe* autoCV Module ===> xgb_CrossValidation with 5 folds:

Best Parameters: {'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 200,
↪'verbosity': 0}

Best CV Score: 0.8464292011990832

Progress: [####################] 100.0%

lgr -- Accuracy: 0.775 / Precision: 0.712 / Recall: 0.646 / Latency: 0.0ms
svm -- Accuracy: 0.747 / Precision: 0.672 / Recall: 0.6 / Latency: 2.0ms
mlp -- Accuracy: 0.787 / Precision: 0.745 / Recall: 0.631 / Latency: 4.1ms
rf -- Accuracy: 0.809 / Precision: 0.83 / Recall: 0.6 / Latency: 37.0ms
ada -- Accuracy: 0.792 / Precision: 0.759 / Recall: 0.631 / Latency: 21.4ms
gb -- Accuracy: 0.815 / Precision: 0.796 / Recall: 0.662 / Latency: 2.0ms
xgb -- Accuracy: 0.815 / Precision: 0.786 / Recall: 0.677 / Latency: 5.0ms
```

8.4 Model selection for a regression problem using autoCV:

Demo Code:

```
import pandas as pd
from dynapipe.autoCV import evaluate_model,dynaClassifier,dynaRegressor
import joblib

from dynapipe.utilis_func import pipeline_splitting_rule, update_parameters,reset_
↪parameters
reset_parameters()

tr_features = pd.read_csv('./data/regression/train_features.csv')
tr_labels = pd.read_csv('./data/regression/train_labels.csv')
val_features = pd.read_csv('./data/regression/val_features.csv')
val_labels = pd.read_csv('./data/regression/val_labels.csv')
te_features = pd.read_csv('./data/regression/test_features.csv')
te_labels = pd.read_csv('./data/regression/test_labels.csv')

reg_cv_demo = dynaRegressor(random_state=13,cv_num = 5)

reg_cv_demo.fit(tr_features,tr_labels)

models = {}

for mdl in ['lr','knn','tree','svm','mlp','rf','gb','ada','xgb','hgboost','huber',
↪'rgcv','cvlasso','sgd']:
    models[mdl] = joblib.load('./pkl/{}_reg_model.pkl'.format(mdl))

for name, mdl in models.items():
    try:
        ml_evl = evaluate_model(model_type = "reg")
        ml_evl.fit(name, mdl, val_features, val_labels)
    except:
        print(f"Failed to load the {mdl}.")
```

Output:

```
Done with the parameters reset.
Now in Progress - Model Selection w/ Cross-validation: Estimate about 0.0337 minutes␣
↪left  [#-------------------] 7.1%

    *DynaPipe* autoCV Module ===> lr model CrossValidation with 5 folds:
Best Parameters: {'normalize': False}

Best CV Score: 0.682929422892965

Now in Progress - Model Selection w/ Cross-validation: Estimate about 0.5549 minutes␣
↪left  [###-----------------] 14.3%

    *DynaPipe* autoCV Module ===> knn model CrossValidation with 5 folds:
Best Parameters: {'algorithm': 'auto', 'n_neighbors': 10, 'weights': 'distance'}

Best CV Score: 0.5277324478219082

Now in Progress - Model Selection w/ Cross-validation: Estimate about 0.2383 minutes␣
↪left  [####----------------] 21.4%

    *DynaPipe* autoCV Module ===> tree model CrossValidation with 5 folds:
Best Parameters: {'max_depth': 5, 'min_samples_leaf': 3, 'splitter': 'best'}
```

(continues on next page)

(continued from previous page)

```
Best CV Score: 0.7704058399460141

Now in Progress - Model Selection w/ Cross-validation: Estimate about 11.0461 minutes␣
↪left  [######--------------] 28.6%

    *DynaPipe* autoCV Module ===> svm model CrossValidation with 5 folds:
Best Parameters: {'C': 1, 'kernel': 'linear'}

Best CV Score: 0.6817778239200576

Now in Progress - Model Selection w/ Cross-validation: Estimate about 20.2113 minutes␣
↪left  [#######-------------] 35.7%

    *DynaPipe* autoCV Module ===> mlp model CrossValidation with 5 folds:
Best Parameters: {'activation': 'identity', 'hidden_layer_sizes': (50,), 'learning_
↪rate': 'constant', 'random_state': 13, 'solver': 'lbfgs'}

Best CV Score: 0.6556246414762388

Now in Progress - Model Selection w/ Cross-validation: Estimate about 3.1693 minutes␣
↪left  [#########-----------] 42.9%

    *DynaPipe* autoCV Module ===> rf model CrossValidation with 5 folds:
Best Parameters: {'max_depth': 8, 'n_estimators': 50}

Best CV Score: 0.8582920563031621

Now in Progress - Model Selection w/ Cross-validation: Estimate about 18.0094 minutes␣
↪left  [##########----------] 50.0%

    *DynaPipe* autoCV Module ===> gb model CrossValidation with 5 folds:
Best Parameters: {'learning_rate': 0.2, 'max_depth': 3, 'n_estimators': 100}

Best CV Score: 0.8794018441486111

Now in Progress - Model Selection w/ Cross-validation: Estimate about 18.7663 minutes␣
↪left  [###########---------] 57.1%

    *DynaPipe* autoCV Module ===> ada model CrossValidation with 5 folds:
Best Parameters: {'learning_rate': 0.3, 'loss': 'linear', 'n_estimators': 150,
↪'random_state': 13}

Best CV Score: 0.8255039215809923

Now in Progress - Model Selection w/ Cross-validation: Estimate about 4.545 minutes␣
↪left  [#############-------] 64.3%

    *DynaPipe* autoCV Module ===> xgb model CrossValidation with 5 folds:
Best Parameters: {'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 300,
↪'verbosity': 0}

Best CV Score: 0.8645505523555148

Now in Progress - Model Selection w/ Cross-validation: Estimate about 1.6471 minutes␣
↪left  [##############------] 71.4%

    *DynaPipe* autoCV Module ===> hgboost model CrossValidation with 5 folds:
```

(continues on next page)

(continued from previous page)

```
Best Parameters: {'learning_rate': 0.2, 'max_depth': 3}

Best CV Score: 0.8490465745463796

Now in Progress - Model Selection w/ Cross-validation: Estimate about 0.0182 minutes␣
↪left  [#################----] 78.6%

    *DynaPipe* autoCV Module ===> huber model CrossValidation with 5 folds:
Best Parameters: {'fit_intercept': False}

Best CV Score: 0.6250877399211718

Now in Progress - Model Selection w/ Cross-validation: Estimate about 0.0024 minutes␣
↪left  [##################---] 85.7%

    *DynaPipe* autoCV Module ===> rgcv model CrossValidation with 5 folds:
Best Parameters: {'fit_intercept': True}

Best CV Score: 0.6814764830347567

Now in Progress - Model Selection w/ Cross-validation: Estimate about 0.011 minutes␣
↪left  [###################-] 92.9%

    *DynaPipe* autoCV Module ===> cvlasso model CrossValidation with 5 folds:
Best Parameters: {'fit_intercept': True}

Best CV Score: 0.6686184981380419

Now in Progress - Model Selection w/ Cross-validation: Estimate about 0.0 minutes␣
↪left  [####################] 100.0%

    *DynaPipe* autoCV Module ===> sgd model CrossValidation with 5 folds:
Best Parameters: {'learning_rate': 'invscaling', 'penalty': 'elasticnet', 'shuffle':␣
↪True}

Best CV Score: -1.445728757185719e+26

lr -- R^2 Score: 0.684 / Mean Absolute Error: 3.674 / Mean Squared Error: 24.037 /␣
↪Root Mean Squared Error: 24.037 / Latency: 2.0s
knn -- R^2 Score: 0.307 / Mean Absolute Error: 4.639 / Mean Squared Error: 52.794 /␣
↪Root Mean Squared Error: 52.794 / Latency: 3.0s
tree -- R^2 Score: 0.671 / Mean Absolute Error: 3.141 / Mean Squared Error: 25.077 /␣
↪Root Mean Squared Error: 25.077 / Latency: 1.0s
svm -- R^2 Score: 0.649 / Mean Absolute Error: 3.466 / Mean Squared Error: 26.746 /␣
↪Root Mean Squared Error: 26.746 / Latency: 7.0s
mlp -- R^2 Score: 0.629 / Mean Absolute Error: 3.56 / Mean Squared Error: 28.244 /␣
↪Root Mean Squared Error: 28.244 / Latency: 4.0s
rf -- R^2 Score: 0.772 / Mean Absolute Error: 2.677 / Mean Squared Error: 17.327 /␣
↪Root Mean Squared Error: 17.327 / Latency: 10.0s
gb -- R^2 Score: 0.775 / Mean Absolute Error: 2.616 / Mean Squared Error: 17.126 /␣
↪Root Mean Squared Error: 17.126 / Latency: 1.0s
ada -- R^2 Score: 0.749 / Mean Absolute Error: 2.933 / Mean Squared Error: 19.09 /␣
↪Root Mean Squared Error: 19.09 / Latency: 18.0s
xgb -- R^2 Score: 0.776 / Mean Absolute Error: 2.66 / Mean Squared Error: 17.02 /␣
↪Root Mean Squared Error: 17.02 / Latency: 5.0s
hgboost -- R^2 Score: 0.758 / Mean Absolute Error: 2.98 / Mean Squared Error: 18.412 /
↪ Root Mean Squared Error: 18.412 / Latency: 9.2s
```

(continues on next page)

(continued from previous page)

```
huber -- R^2 Score: 0.613 / Mean Absolute Error: 3.63 / Mean Squared Error: 29.476 /␣
↪Root Mean Squared Error: 29.476 / Latency: 4.0s
rgcv -- R^2 Score: 0.672 / Mean Absolute Error: 3.757 / Mean Squared Error: 24.983 /␣
↪Root Mean Squared Error: 24.983 / Latency: 3.0s
cvlasso -- R^2 Score: 0.661 / Mean Absolute Error: 3.741 / Mean Squared Error: 25.821␣
↪/ Root Mean Squared Error: 25.821 / Latency: 4.0s
sgd -- R^2 Score: -7.6819521340367e+26 / Mean Absolute Error: 239048363331832.62 /␣
↪Mean Squared Error: 5.849722584020232e+28 / Root Mean Squared Error: 5.
↪849722584020232e+28 / Latency: 1.0s
```

8.5 Custom estimators & parameters setting for for autoCV:

Currently, there're 3 methods in *utilis_fun* module - *reset_parameters*, *update_parameters*, and *export_parameters*.

- *update_parameters* method is used to modify the default parameter settings for models selection module (autoCV).

  i.e. When you want to modify the support vector machine classifier, with new penalty "C" and "kernel" values, the code line below will achieve that.

```
update_parameters(mode = "cls", estimator_name = "svm", C=[0.1,0.2],kernel=["linear"])
```

- *export_parameters* method can help you export the currnt default parameter settings as 2 csv files named "exported_cls_parameters.csv" and "exported_reg_parameters.csv". You can find them in the *./exported* folder of you current work dictionary.

```
export_parameters()
```

- *reset_parameters* method can reset the default parameter settings to the package's original default settings. Just add this code line will work:

```
reset_parameters()
```

8.6 Build Pipeline Cluster Traveral Experiments using autoPipe:

Demo Code:

```
import pandas as pd
from dynapipe.autoPipe import autoPipe
from dynapipe.funcPP import PPtools
from dynapipe.autoPP import dynaPreprocessing
from dynapipe.autoFS import dynaFS_clf
from dynapipe.autoCV import evaluate_model,dynaClassifier

df = pd.read_csv('./data/preprocessing/breast_cancer.csv')

pipe = autoPipe(
[("autoPP",dynaPreprocessing(custom_parameters = None, label_col = 'diagnosis', model_
↪type = "cls")),
("datasets_splitting",pipeline_splitting_rule(val_size = 0.2, test_size = 0.2, random_
↪state = 13)),
```

(continues on next page)

(continued from previous page)

```
("autoFS",dynaFS_clf(fs_num = 5, random_state=13, cv = 5, in_pipeline = True, input_
↪from_file = False)),
("autoCV",dynaClassifier(random_state = 13,cv_num = 5,in_pipeline = True, input_from_
↪file = False)),
("model_evaluate",evaluate_model(model_type = "cls"))])

dyna_report= pipe.fit(df)[4]
dyna_report.head(5)
```

Output:

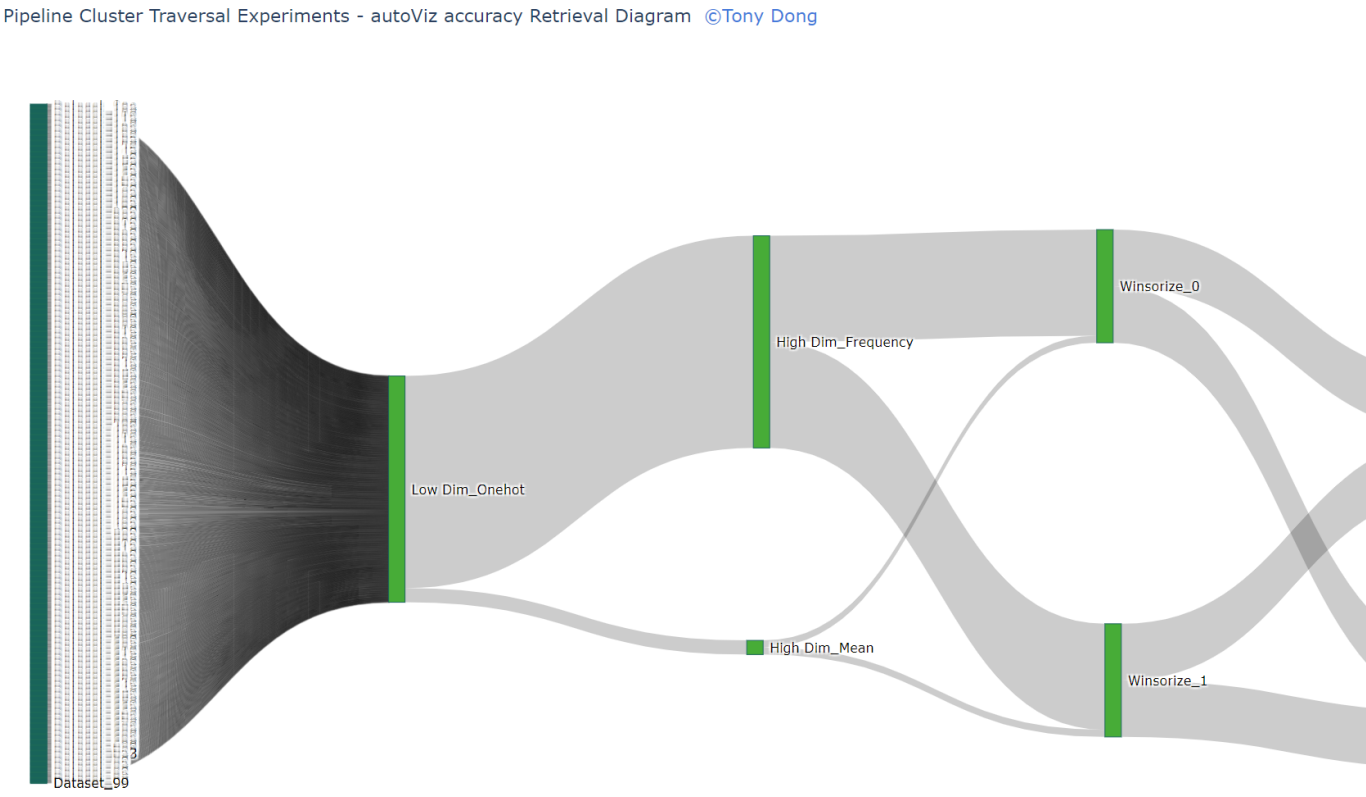
```
      Dataset    Model_Name    Best_Parameters         Accuracy       Precision
↪Recall  Latency
1     Dataset_0       svm     [('C', 0.1), ('kernel', 'linear')]        0.930 0.889 0.
↪96 3.0
6     Dataset_0       xgb     [('learning_rate', 1), ('max_depth', 2), ('n_estimators
↪', 50), ('random_state', 13)]    0.912   0.955   0.84    2.0
40    Dataset_5       gb      [('learning_rate', 1), ('max_depth', 2), ('n_estimators
↪', 50), ('random_state', 13)]    0.895   0.913   0.84    2.0
31    Dataset_4       rf      [('max_depth', 2), ('n_estimators', 50), ('random_state
↪', 13)]  0.877   0.821   0.92    12.0
51    Dataset_7       mlp     [('activation', 'relu'), ('hidden_layer_sizes', (10,)),
↪('learning_rate', 'constant'), ('random_state', 13), ('solver', 'sgd')] 0.772   0.
↪875   0.56    4.0
```

8.7 Pipeline Cluster Traversal Experiments Model Retrieval Diagram using autoViz:

Demo Code:

```
from dynapipe.autoViz import autoViz
viz = autoViz(preprocess_dict=DICT_PREPROCESSING,report=dyna_report)
viz.clf_model_retrieval(metrics='accuracy')
```

Output:


Pipeline Cluster Traversal Experiments - autoViz accuracy Retrieval Diagram ©Tony Dong
Dataset_99
Low Dim_Onehot
High Dim_Frequency
High Dim_Mean
Winsorize_0
Winsorize_1

CHAPTER 9

Contributers

9.1 Original Author

- Tony Dong <https://www.linkedin.com/in/lei-tony-dong/>

9.2 Contributors

- Send email to <tonyleidong@gmail.com> to be one of us.

CHAPTER 10

History

10.1 0.2.2 (2020-8-7)

- Add estimators: HuberRegressor, RidgeCV, LassoCV, SGDRegressor, and HistGradientBoostingRegressor
- Modify parameters.json, and reset_parameters.json for the added estimators
- Add autoViz for classification problem model retrieval diagram

10.2 0.2.1 (2020-7-31)

- Bug fix and stable version with pipeline modules

10.3 0.1.4 (2020-7-30)

- Add autoPP & autoPipe modules

10.4 0.1.3 (2020-07-21)

- Add online documentation page: https://dynamic-pipeline.readthedocs.io

10.5 0.1.0 (2020-07-21)

- Debug in the estimatorCV.py

10.6 0.0.11 (2020-07-21)

- Add JSON file as the repository for the estimators' parameters
- Added functions reset_parameters(), export_parameters(), and reset_parameters() to manipulate the default parameters for estimatorCV.py
- Update the README.md file
- Debug in the autoCV.py

10.7 0.0.8 (2020-07-18)

- First release on PyPI.

CHAPTER 11

Report Issues

Report Issues at https://github.com/tonyleidong/dynapipe/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

CHAPTER 12

Indices and tables

- genindex
- modindex
- search

Bibliography

[Example] https://dynamic-pipeline.readthedocs.io/en/latest/demos.html#feature-preprocessing-for-a-regression-problem-using-autop

[Example] https://dynamic-pipeline.readthedocs.io/en/latest/demos.html#build-pipeline-cluster-traveral-experiments-using-autopipe

[Example] https://dynamic-pipeline.readthedocs.io/en/latest/demos.html#features-selection-for-a-regression-problem-using-autoFS

[Example]

[Example]

[Example] https://dynamic-pipeline.readthedocs.io/en/latest/demos.html#model-selection-for-a-classification-problem-using-autocv

[Example] https://dynamic-pipeline.readthedocs.io/en/latest/demos.html#model-selection-for-a-regression-problem-using-autocv

[Example] https://dynamic-pipeline.readthedocs.io/en/latest/demos.html#build-pipeline-cluster-traveral-experiments-using-autopipe

[Example]

[Example]

[Example] https://dynamic-pipeline.readthedocs.io/en/latest/demos.html#build-pipeline-cluster-traveral-experiments-using-autopipe

[Example] https://dynamic-pipeline.readthedocs.io/en/latest/demos.html#pipeline-cluster-traversal-experiments-model-retrieval-diagra

Python Module Index

d

Index