

# Automated Machine Learning (AutoML) in Insurance

Aug. 4, 2022

Presenter: Panyi Dong



UNIVERSITY OF  
**ILLINOIS**  
URBANA-CHAMPAIGN

# 1. Introduction to AutoML

**Machine learning (ML)** is a field of **Artificial Intelligence (AI)** and a field of inquiry devoted to understanding and building methods that 'learn', that is, methods that **leverage data** to improve performance on some set of tasks. [1]

Common application scenarios:

- (1) Self Driving Cars
- (2) Recommending Systems
- (3) Automated Translation
- (4) ....

For Insurance:

- (1) Predict future claim frequency/severity
- (2) Fraud prevention
- (3) ...



Fig. Self-Driving Cars [2]

# 1. Introduction to AutoML

However, ML tasks can be **experience-dependent** and **heavy manual work**.

Given the nature of ML algorithms, which are **data-driven**, selection of models and hyperparameters are critical for each dataset, and **no universal solution** exists.

Furthermore, industrial datasets usually are not well-formatted or well-organized, and problems like **missing values, irrelevant features, imbalance distributions** exists.

It's difficult for those who have no previous experience/knowledge to gain hands-on experience.

# 1. Introduction to AutoML

Automated Machine Learning (AutoML) is one of the solutions.

AutoML tries to automatically select a ML model and tuning for optimal hyperparameters, so that “non-expert users” can apply ML to their application scenarios more effectively and “achieve improved performance”. [3]

```
save_path = 'agModels-predictClass' # specifies folder to store trained models
predictor = TabularPredictor(label=label, path=save_path).fit(train_data)
```

Fig. AutoGluon [4]

```
import autosklearn.classification
cls = autosklearn.classification.AutoSklearnClassifier()
cls.fit(X_train, y_train)
predictions = cls.predict(X_test)
```

Fig. auto-sklearn [5]

```
>>> import autoPyTorch
>>> cls = autoPyTorch.api.tabular_classification.TabularClassificationTask()
>>> cls.search(X_train, y_train)
>>> predictions = cls.predict(X_test)
```

Fig. Auto-PyTorch [ 6]

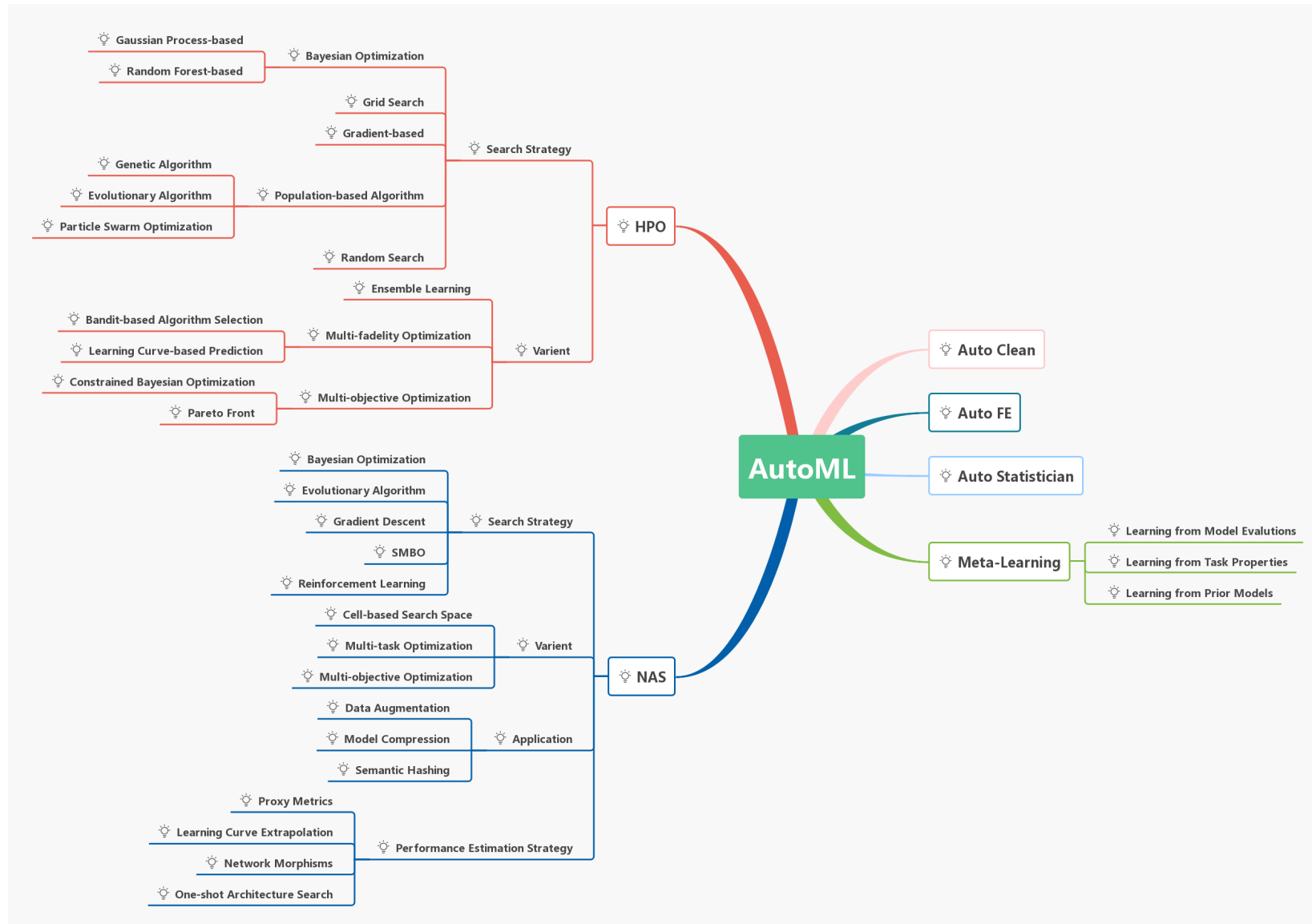


Fig. AutoML research mainstream [7]

# 1. Introduction to AutoML

Key criteria of AutoML:

(1) **Better Performance**

What's a proper search space; How the model/hyperparameter evaluated;  
Better model search algorithms, hyperparameter optimization algorithms; ...

(2) **Higher Efficiency**

AutoML usually evaluate by training multiple models, how to more efficiently assess  
(or sometimes with limited computation resources)

(3) **Ease of use, Robustness**

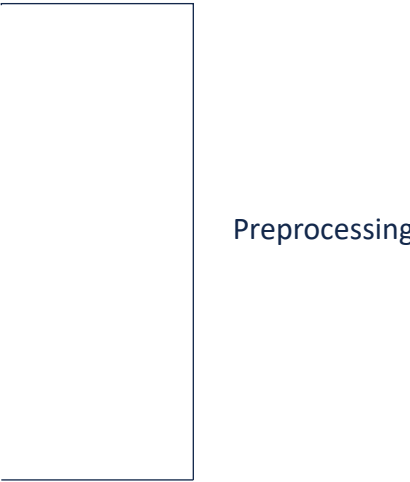
The goal is to ease expertise required, a robust AutoML for all possible scenarios

# 1. Introduction to AutoML

Our AutoML pipeline:

- (1) Complete, fully functional processing and model tuning
- (2) Special treatment for imbalanced datasets
- (3) Record training process and store the optimal pipeline for continued applications

## 2. Components & Pipeline

1. Data Encoding
  2. Data Imputation
  3. Data Balancing
  4. Data Scaling
  5. Feature Selection
  6. Classification/Regression Models
  7. Model Selection and Hyperparameter Optimization
- 
- A diagram consisting of a vertical line on the right side of steps 1 through 5, with horizontal lines extending from the right of each step to this vertical line. The word 'Preprocessing' is placed to the right of this vertical line, centered between steps 3 and 4.
- Preprocessing



## 2. Components & Pipeline

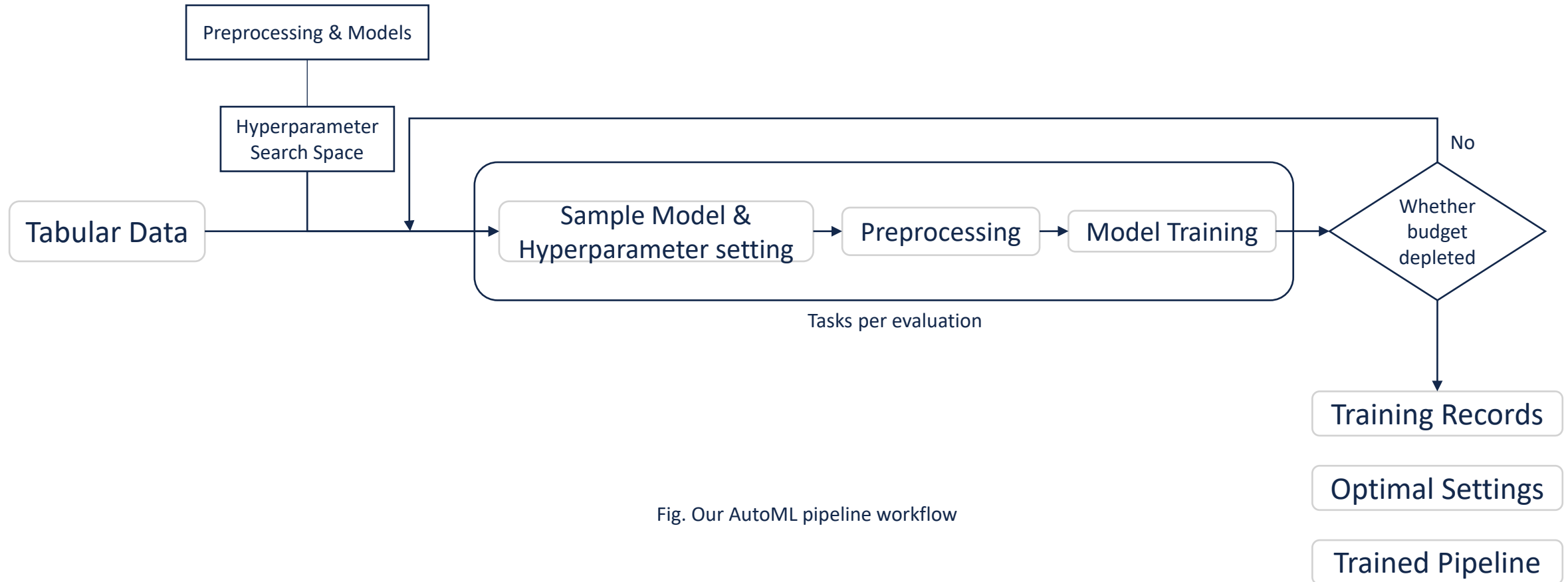


Fig. Our AutoML pipeline workflow

## 2.1. Data Encoding

Real-life datasets sometimes contain features of **string type**.

age	sex	bmi	children	smoker	region	expenses
19	female	27.9	0	yes	southwest	16884.92
18	male	33.8	1	no	southeast	1725.55
28	male	33	3	no	southeast	4449.46
33	male	22.7	0	no	northwest	21984.47
32	male	28.9	0	no	northwest	3866.86
31	female	25.7	0	no	southeast	3756.62
46	female	33.4	1	no	southeast	8240.59
37	female	27.7	3	no	northwest	7281.51
37	male	29.8	2	no	northeast	6406.41

Convert unique strings  
to numerical types

age	sex	bmi	children	smoker	region	expenses
19	0	27.9	0	1	3	16884.92
18	1	33.8	1	0	2	1725.55
28	1	33	3	0	2	4449.46
33	1	22.7	0	0	1	21984.47
32	1	28.9	0	0	1	3866.86
31	0	25.7	0	0	2	3756.62
46	0	33.4	1	0	2	8240.59
37	0	27.7	3	0	1	7281.51
37	1	29.8	2	0	0	6406.41

One-hot encoding

age	bmi	children	expenses	sex_female	sex_male	smoker_no	smoker_yes	region_northeast	region_northwest	region_southeast	region_southwest
19	27.9	0	16884.92	1	0	0	1	0	0	0	1
18	33.8	1	1725.55	0	1	1	0	0	0	1	0
28	33	3	4449.46	0	1	1	0	0	0	1	0
33	22.7	0	21984.47	0	1	1	0	0	1	0	0
32	28.9	0	3866.86	0	1	1	0	0	1	0	0
31	25.7	0	3756.62	1	0	1	0	0	0	1	0
46	33.4	1	8240.59	1	0	1	0	0	0	1	0
37	27.7	3	7281.51	1	0	1	0	0	1	0	0
37	29.8	2	6406.41	0	1	1	0	1	0	0	0

Majority of common ML methods in Python does not support string as inputs, encoding becomes necessary.

## 2.2. Data Imputation

Some of datasets contains **missing values** where true values can not be traced.

Causes: Incomplete data entry, Save malfunction, ...

Common solutions:

(1) Delete missing values.

**Advantage:** all remaining values are true values.

**Drawback:** not feasible when data size is small.

(2) Impute the missing values with estimated values.

**Advantage:** best utilize all available information.

**Drawback:** imputed values may not accurately describe the missing values, time-consuming, ...

## 2.2. Data Imputation

Incorporated imputation methods:

### 2.2.1. Simple Imputation

Most common and efficient imputation methods. Use non-missing summary statistics (mean/median/...) to fill the missing values.

**Drawback:** Indifferent to the variation of other information.

### 2.2.2. Joint Imputation

Assume all features follow some multi-variate distributions, whose parameters (e.g., mean vector  $\mu$  and covariance matrix  $\Sigma$  in multi-normal distributions) are determined by observed values.

**Drawback:** Ideal assumption of joint distribution usually do not exist.

## 2.2. Data Imputation

Incorporated imputation methods:

### 2.2.3. Expectation Maximization (EM) [8]

Utilize the classical EM algorithm, where the E step tries to create a function of log-likelihood, and M step tries to maximize the likelihood.

**Drawback:** underestimate standard error of imputed values.

### 2.2.4. KNN Imputation [9]

### 2.2.5. Miss Forest Imputation [9]

2.2.4, 2.2.5 builds corresponding models (k Nearest Neighbors for 2.2.4 and Random Forest for 2.2.5) on observed values and predictions as imputation values.

## 2.2. Data Imputation

Incorporated imputation methods:

### 2.2.6. Multiple Imputation by Chained Equations (MICE) [10]

One of most popular Multiple Imputation methods, multiple copies datasets are created and each missing values are filled with multiple cycles of predictions of model fitted.

**Drawback:** Time-consuming for not small datasets to converge.

### 2.2.7. Generative Adversarial Imputation Nets (GAIN) [11]

Utilize the Generative Adversarial Network (GAN) to train a Generator ( $G$ ) and Discriminator ( $D$ ) where  $G$  tries to generate values for missing positions and  $D$  tries to distinguish whether the generated values are valid.

**Drawback:** Large data size required for neural network training, and time-consuming.

## 2.3. Data Balancing

Some of datasets exhibits **imbalanced nature** where majority values are the same (e.g., 99% of policyholders do not report claims).

Traditional ML models evaluated by classical metrics may not be the ideal solution by the business perspective.

Common solutions:

- (1) Increase the weights of minority class.
- (2) Over-sampling on minority class/Down-sampling on majority class. [12]
- (3) Create a model ensemble (multiple models weighted on prediction).
- (4) Special loss metrics for gradient-based models.
- (5) ...

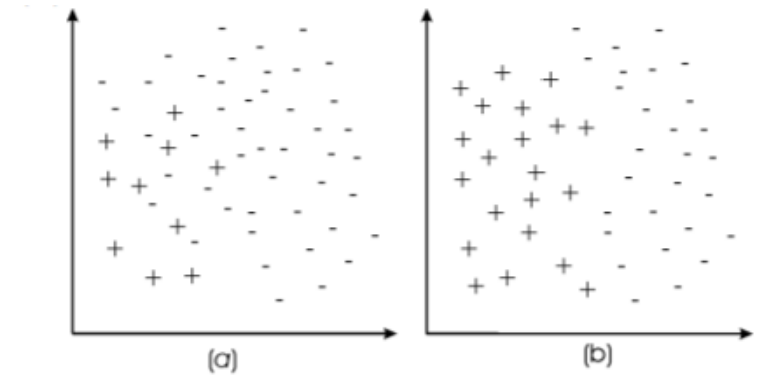


Fig. Spatial illustration of unbalanced datasets (a) and balanced datasets (b). [12]

## 2.3. Data Balancing

Incorporated balancing methods:  
(focusing on over-sampling/down-sampling at this step)

### 2.3.1. Simple Random Over-Sampling

### 2.3.2. Simple Random Down-Sampling

Method 2.3.1 and 2.3.2 simply randomly copy minority class entries/eliminate majority class entries.

### 2.3.3. Tomek Link [13]

An Under-Sampling method.  
Remove noise majority class around decision boundaries.

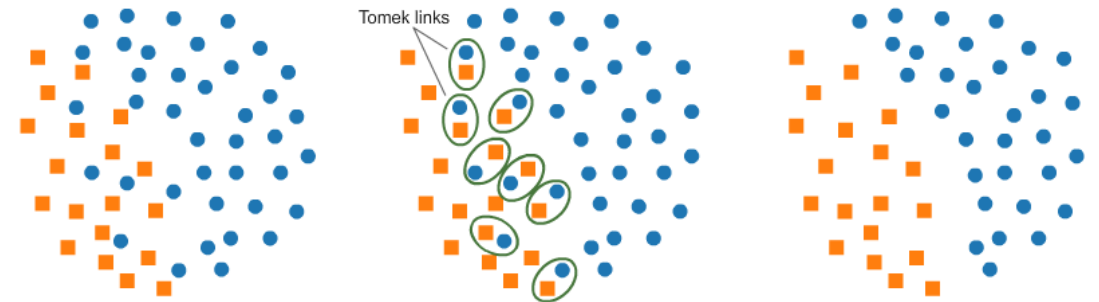


Fig. Illustration of Tomek Link [14]



## 2.3. Data Balancing

Incorporated balancing methods:  
(focusing on over-sampling/down-sampling at this step)

### 2.3.4. Edited Nearest Neighbors (ENN) [15]

An Under-Sampling method.

Remove majority class observations which disagree with predictions from kNN model.

### 2.3.5. Condensed Nearest Neighbors (CNN) [17]

An Under-Sampling method.

Select a subset of observations  $\hat{E} \subseteq E$  where  $\hat{E}$  can predict all observations correctly using 1-NN.

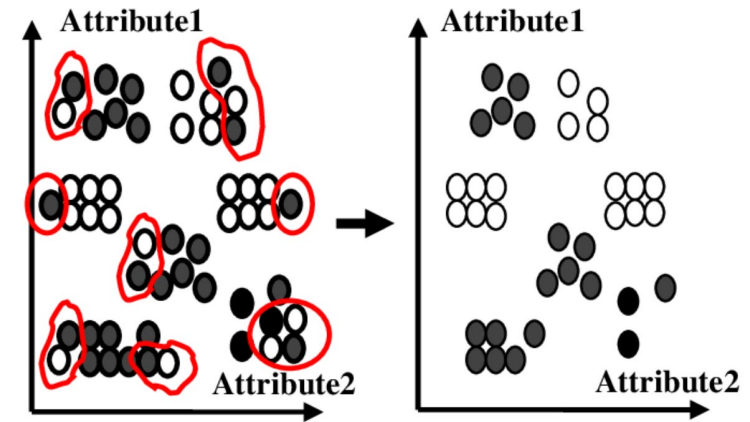


Fig. Illustration Edited Nearest Neighbor [16]

## 2.3. Data Balancing

Incorporated balancing methods:  
(focusing on over-sampling/down-sampling at this step)

### 2.3.6. Synthetic Minority Over-Sampling Technique (Smote) [18]

A common Over-Sampling method.

Try to generate more realistic synthetic minority observations by interpolation between minority observations.

### 2.3.7. One Sided Selection (OSS): Tomek Link + CNN

### 2.3.8. CNN\_Tomek Link: CNN + Tomek Link

### 2.3.9. Smote\_Tomek Link: Smote + Tomek Link

### 2.3.10. Smote\_ENN: Smote + ENN

Combination of two-step balancing methods, take advantages of both steps.

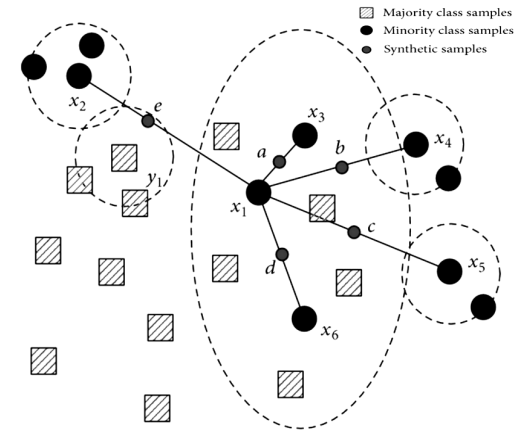


Fig. Illustration of Smote [19]

## 2.4. Data Scaling

Scaling intends to transform data to fit specific scale so it may converge more easily.

Incorporated Scaling methods:

### 2.4.1. Standardize

All features are standardized using  $(x - \bar{x})/\sigma_x$  where  $\bar{x}$  is the mean,  $\sigma_x$  is the standard deviation.

### 2.4.2. Normalize

Feature are normalized using  $x/x_{max}$  into unit scale.

### 2.4.3. Robust Scaling

Feature are scaled by range of two feature quantiles.  
(e.g., scaled by the range between 25<sup>th</sup> quantile and 75<sup>th</sup> quantile)

## 2.4. Data Scaling

Incorporated Scaling methods:

### 2.4.4. Min-Max Scaling

Feature are scaled by min/max values to  $[0,1]$  range.

### 2.4.5. Power Transformer

Apply a power transformation (Box-Cox/Yeo-Johnson transformation) to each feature to shape data more normal-distributed.

### 2.4.6. Winsorization

Cap features at certain quantile to avoid extreme values (outliers).

## 2.5. Feature Selection

Modern datasets have hundreds of (or even more) features with exploding size.

However, some of them may be reductant or irrelevant, which may decrease the model performance while still takes long training time. One of the solution is using **feature selection** to select only subset of features for model training.

Common categories of feature selection [20]:

1. **Filter:** use statistical analysis in only feature space
2. **Wrapper:** train a model on feature subsets and use performance as selection criteria.
3. **Embedded:** embed feature importance into model training and use as selection criteria.
4. **Hybrid:** combine filter and wrapper for feature selection.

# 2.5. Feature Selection

Incorporated Feature Selection methods:

## 2.5.1. Feature Filter

Feature filter select relevant features by scoring features and selecting highest score features.

(1) Pearson Correlation Coefficient

$$R(i) = \frac{Cov(X_i, Y)}{\sqrt{Var(X_i)Var(Y)}}$$

(2) Mutual Information  $I(X, Y) = H(Y) - H(Y|X)$

using Shannon Entropy

$$H(Y) = - \sum_y p(y) \log(p(y)), H(Y|X) = - \sum_x \sum_y p(x, y) \log(p(y|x))$$

# 2.5. Feature Selection

Incorporated Feature Selection methods:

## 2.5.2. Adaptive Sequential Forward Floating Search (ASFFS) [21]

ASFFS is a variation of sequential feature selection (SFS) method to deal with feature correlation.

ASFFS iteratively performs forward phase and backward until target reached where forward phase adds features, backward phase removes features.

ASFFS will adaptively change number of features consider at a time.

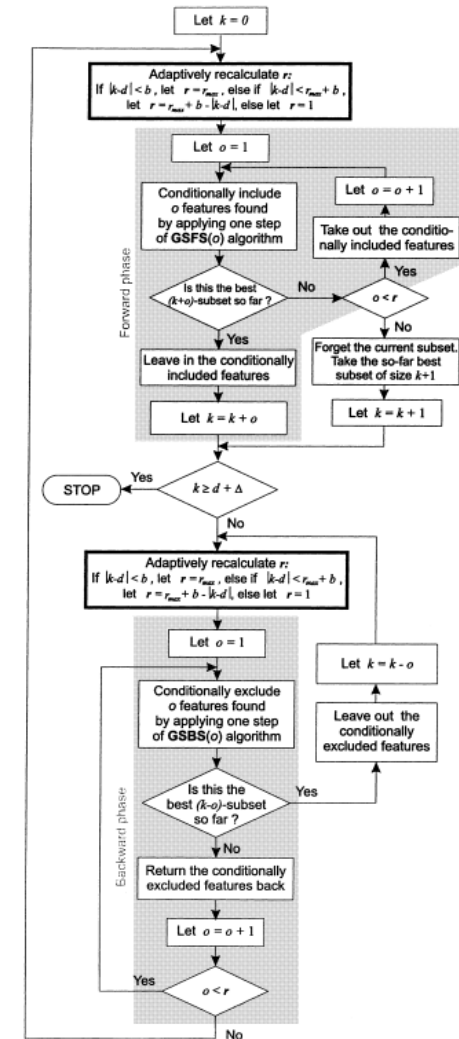


Fig. Workflow of ASFFS [12]

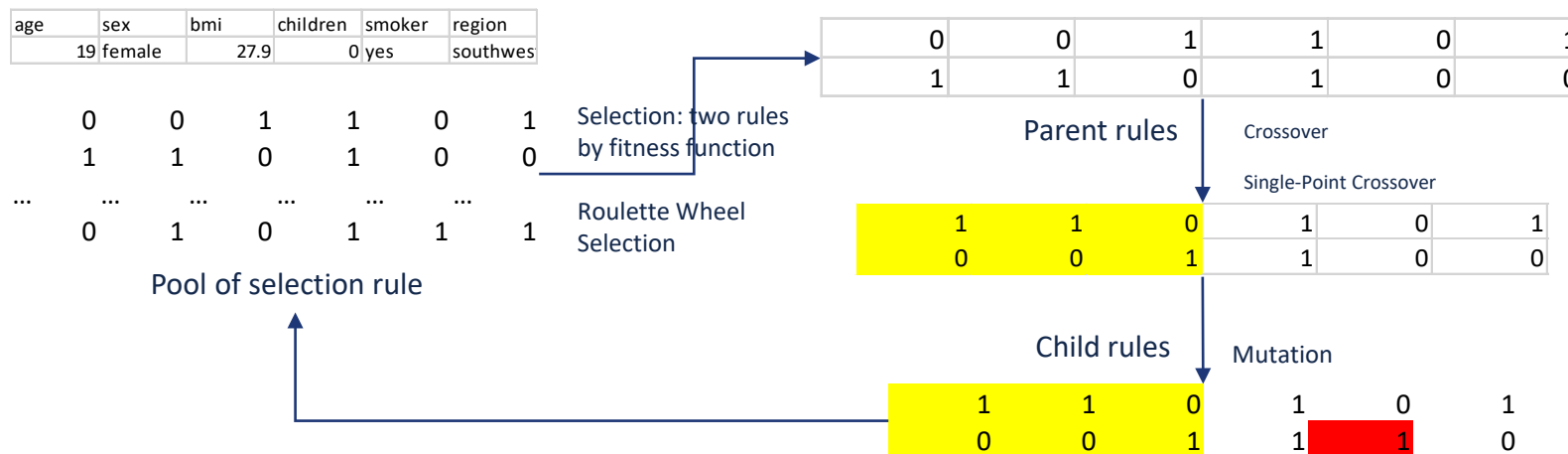
# 2.5. Feature Selection

Incorporated Feature Selection methods:

## 2.5.3. Genetic Algorithm (GA) [22]

GA is a popular algorithm inspired by natural DNA replication/mutation.

GA consists of selection, crossover and mutation phase at each generation. With generations of offspring, best feature selection rules will be selected.



During the process, all rules are evaluated by fitness function with their performance and assigned a possibility of selection.



## 2.5. Feature Selection

Incorporated Feature Selection methods:

### 2.5.4. minimal-Redundancy-Maximal-Relevance (mRMR) [23] [24] [25]

mRMR is a filter method, and a variation of SFS to both maximize relevance and minimize redundancy among selected features. Both redundancy and relevance are defined as mutual information.

### 2.5.5. Copula-based Feature Selection (CBFS) [26] [27]

CBFS is a filter method where the data is described using copulas, which is a popular method to describe multivariate correlation.

Features are scored by mutual information of copula values.

# 2.6.1. Classification Models

Incorporated Classification models:

2.6.1.1.      **Logistic Regression**

2.6.1.2.      **Adaboost [28]**

2.6.1.3.      **Hist Gradient Boosting**

2.6.1.4.      **Linear Support Vector Machine (SVM) [30]**

2.6.1.5.      **Kernel Support Vector Machine (SVM) [31]**

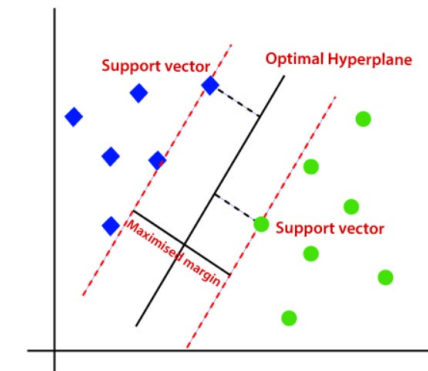


Fig. Support Vector Machine [32]

# 2.6.1. Classification Models

Incorporated Classification models:

2.6.1.6. Decision Tree

2.6.1.7. Extra Trees

2.6.1.8. Random Forest

2.6.1.9. K Nearest Neighbors

2.6.1.10. SGD

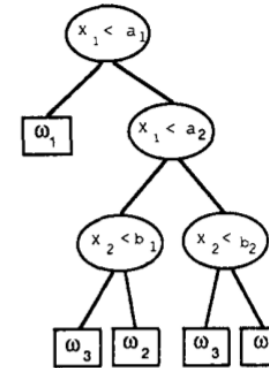


Fig. Decision Tree Split [29]

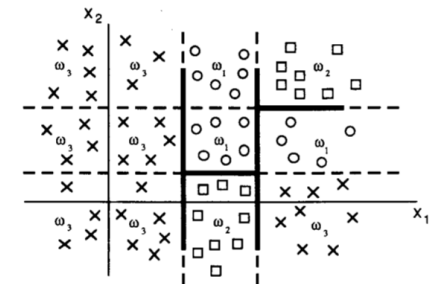


Fig. Split On Data Points [29]

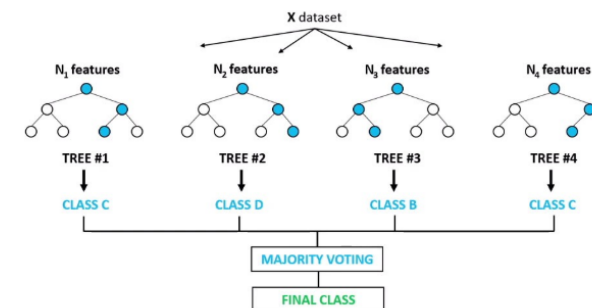


Fig. Random Forest [41]

# 2.6.1. Classification Models

Incorporated Classification models:

2.6.1.11. **Linear Discriminant Analysis (LDA)**

2.6.1.12. **Quadratic Discriminant Analysis (QDA)**

2.6.1.13. **Bernoulli Naïve Bayes**

2.6.1.14. **Gaussian Naïve Bayes**

2.6.1.15. **Multinomial Naïve Bayes**

Fig. Linear Discriminant Analysis

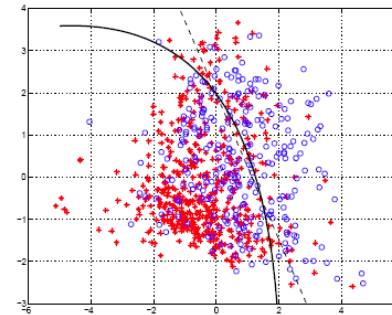
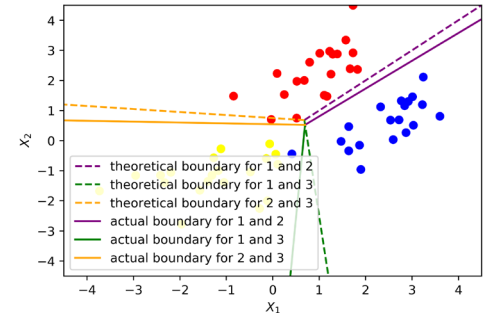


Fig. Quadratic Discriminant Analysis [42]

# 2.6.1. Classification Models

Incorporated Classification models:

2.6.1.16.      **Multi-Layer Perception (MLP)**

2.6.1.17.      **Passive Aggressive**

2.6.1.18.      **Light Gradient Boosting Machine (LightGBM) [36]**

2.6.1.19.      **Extreme Gradient Boosting (XGBoost) [37]**

2.6.1.20.      **Generalized Additive Models (GAM) [38]**

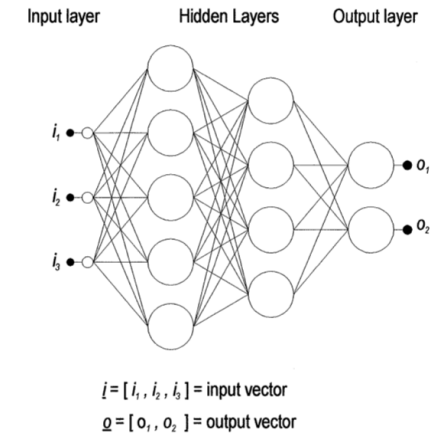


Fig. Multi-Layer Perceptron [33]

## 2.6.2. Regression Models

Incorporated Regression models:

2.6.2.1. Linear Regression

2.6.2.2. Lasso Regression

2.6.2.3. Ridge Regression

2.6.2.4. ElasticNet

2.6.2.5. Bayesian Ridge Regression [34]

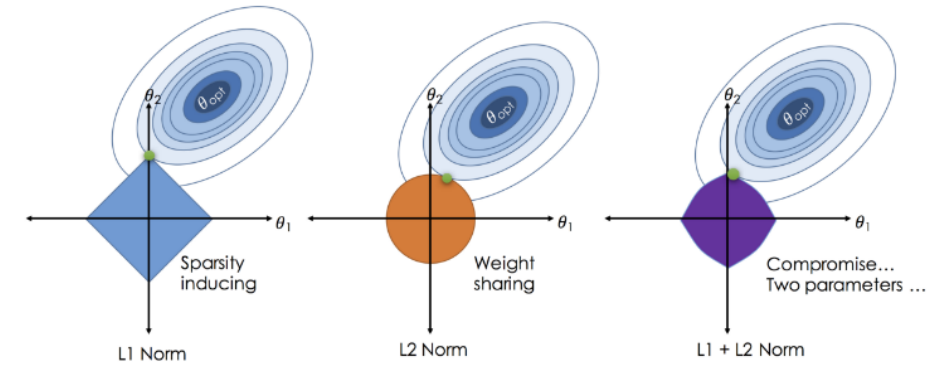


Fig. ElasticNet [43]

## 2.6.2. Regression Models

Incorporated Regression models:

2.6.2.6. **Adaboost**

2.6.2.7. **ARD Regression**

2.6.2.8. **Decision Tree**

2.6.2.9. **Extra Trees**

2.6.2.10. **Random Forest**

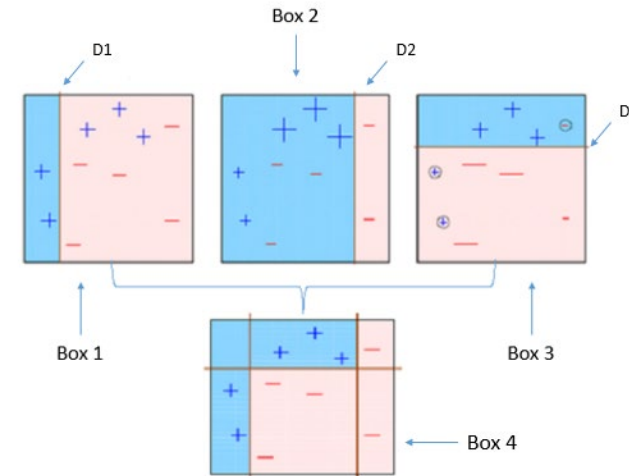


Fig. Adaboost [44]

## 2.6.2. Regression Models

Incorporated Regression models:

- 2.6.2.11. Gaussian Process [35]
- 2.6.2.12. Hist Gradient Boosting
- 2.6.2.13. k Nearest Neighbors
- 2.6.2.14. Linear Support Vector Machine (SVM)
- 2.6.2.15. Multi-Layer Perceptron (MLP)

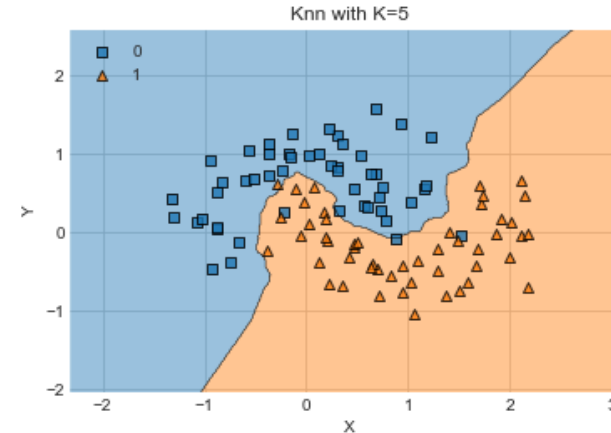


Fig. kNN [45]



## 2.6.2. Regression Models

Incorporated Regression models:

**2.6.2.16. SGD**

**2.6.1.17. Light Gradient Boosting Machine (LightGBM) [36]**

**2.6.1.18. Extreme Gradient Boosting (XGBoost) [37]**

**2.6.1.19. Generalized Additive Models (GAM) [38]**

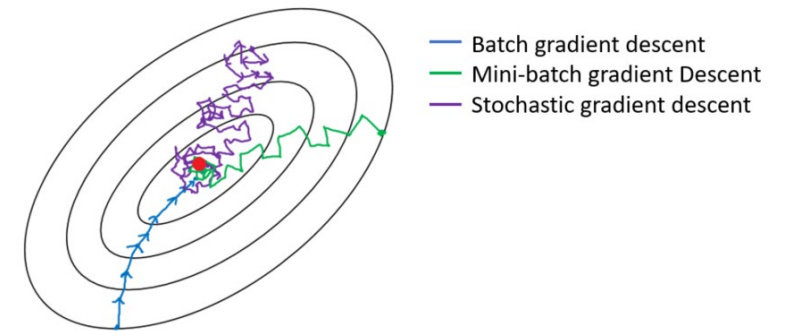


Fig. Different Gradient Descent [46]

## 2.7. Model Selection & Hyperparameter Optimization



To connect all preprocessing methods and models with hyperparameter space, we use **ray.tune** [47] for model selection and hyperparameter optimization.

**ray.tune** is a scalable Python package to conduct experiments on hyperparameter tuning with all common ML model structures (**scikit-learn** [39], **TensorFlow** [48], **PyTorch** [49], ...) with search algorithms like **Optuna** [50], **HyperOpt** [51], ...

# 2.7. Model Selection & Hyperparameter Optimization

```
== Status ==
Current time: 2022-07-31 15:43:49 (running for 00:03:03.19)
Memory usage on this node: 3.4/11.6 GiB
Using FIFO scheduling algorithm.
Resources requested: 0/0 CPUs, 0/0 GPUs, 0.0/5.36 GiB heap, 0.0/2.68 GiB objects (0.0/1.0 accelerator type(s) in use).
Current best trial: 56ae251f with loss=-0.8632478632478633 and parameters={'task_type': 'tabular_classification'}
Result logdir: /home/panyi/Git Repo/My AutoML/tmp/heart_model
Number of trials: 64/64 (64 TERMINATED)
```

Trial name	status	loc	task_type	iter	total time (s)	fitted_model	training_status	loss
TabularObjective_0e04be71	TERMINATED	192.168.1.89:13207	tabular_classification	4	1.97825	Ltbsvm_svc	fitted	-0.598291
TabularObjective_0f981641	TERMINATED	192.168.1.89:13253	tabular_classification	4	3.27871	GaussianNB	fitted	-0.769231
TabularObjective_0fa7b272	TERMINATED	192.168.1.89:13255	tabular_classification	4	20.4705	AdaBoostClassifier	fitted	-0.837607
TabularObjective_0fc002e1	TERMINATED	192.168.1.89:13259	tabular_classification	4	2.32355	QDA	fitted	-0.760684
TabularObjective_0fdaf193	TERMINATED	192.168.1.89:13262	tabular_classification	4	3.57492	BernoulliNB	fitted	-0.589744
TabularObjective_0feb97b1	TERMINATED	192.168.1.89:13264	tabular_classification	4	144.201	Ltbsvm_svc	fitted	-0.435897
TabularObjective_1017578c	TERMINATED	192.168.1.89:13266	tabular_classification	4	18.5566	HistGradientBoostingClassifier	fitted	-0.846154
TabularObjective_1035621f	TERMINATED	192.168.1.89:13284	tabular_classification	4	1.58028	QDA	fitted	-0.623932
TabularObjective_105bf1b1	TERMINATED	192.168.1.89:13207	tabular_classification	4	2.67778	LDA	fitted	-0.846154
TabularObjective_10f02794	TERMINATED	192.168.1.89:13207	tabular_classification	4	8.94002	RandomForest	fitted	-0.57265
TabularObjective_13e17633	TERMINATED	192.168.1.89:13284	tabular_classification	4	162.87	GradientBoostingClassifier	fitted	-0.632479
TabularObjective_1422f777	TERMINATED	192.168.1.89:13259	tabular_classification	4	12.2744	ExtraTreesClassifier	fitted	-0.786325
TabularObjective_144eddbb	TERMINATED	192.168.1.89:13253	tabular_classification	4	9.62179	MLPClassifier	fitted	-0.769231
TabularObjective_14c831ff	TERMINATED	192.168.1.89:13262	tabular_classification	4	6.38417	MultinomialNB	fitted	-0.777778
TabularObjective_1513c554	TERMINATED	192.168.1.89:13262	tabular_classification	4	3.80287	MLPClassifier	fitted	-0.794872
TabularObjective_18ec30f5	TERMINATED	192.168.1.89:13207	tabular_classification	4	6.19195	GaussianNB	fitted	-0.803419
TabularObjective_1944fd72	TERMINATED	192.168.1.89:13253	tabular_classification	4	5.22084	HistGradientBoostingClassifier	fitted	-0.777778
TabularObjective_1ab0c62a	TERMINATED	192.168.1.89:13262	tabular_classification	4	8.71007	ExtraTreesClassifier	fitted	-0.769231
TabularObjective_1b71772e	TERMINATED	192.168.1.89:13259	tabular_classification	4	4.5688	LogisticRegression	fitted	-0.846154
TabularObjective_1ba85298	TERMINATED	192.168.1.89:13207	tabular_classification	4	158.035	Ltbsvm_svc	fitted	-0.589744
TabularObjective_1cf86a4a	TERMINATED	192.168.1.89:13266	tabular_classification	4	6.16606	LDA	fitted	-0.803419
TabularObjective_1e4842c7	TERMINATED	192.168.1.89:13253	tabular_classification	4	67.4649	HistGradientBoostingClassifier	fitted	-0.794872
TabularObjective_1eb61ad9	TERMINATED	192.168.1.89:13259	tabular_classification	4	2.27325	LDA	fitted	-0.794872
TabularObjective_1f14194c	TERMINATED	192.168.1.89:13255	tabular_classification	4	5.0608	LogisticRegression	fitted	-0.846154
TabularObjective_1f6694d7	TERMINATED	192.168.1.89:13259	tabular_classification	4	2.79101	DecisionTree	fitted	-0.786325
TabularObjective_20f36988	TERMINATED	192.168.1.89:13262	tabular_classification	4	2.55617	LogisticRegression	fitted	-0.794872
TabularObjective_215a8d15	TERMINATED	192.168.1.89:13259	tabular_classification	4	3.24688	PassiveAggressive	fitted	-0.752137
TabularObjective_22b3cf74	TERMINATED	192.168.1.89:13262	tabular_classification	4	2.40195	SGD	fitted	-0.649573
TabularObjective_232c3cae	TERMINATED	192.168.1.89:13255	tabular_classification	4	84.4309	KNeighborsClassifier	fitted	-0.649573
TabularObjective_2372f615	TERMINATED	192.168.1.89:13266	tabular_classification	4	2.80615	LogisticRegression	fitted	-0.811966
TabularObjective_23c81af2	TERMINATED	192.168.1.89:13262	tabular_classification	4	2.77573	LDA	fitted	-0.615385
TabularObjective_25128f10	TERMINATED	192.168.1.89:13259	tabular_classification	4	5.02668	LogisticRegression	fitted	-0.846154
TabularObjective_2598d4a8	TERMINATED	192.168.1.89:13266	tabular_classification	4	61.6135	LogisticRegression	fitted	-0.82906
TabularObjective_26030ca9	TERMINATED	192.168.1.89:13262	tabular_classification	4	8.97118	AdaBoostClassifier	fitted	-0.777778
TabularObjective_274d7be4	TERMINATED	192.168.1.89:13259	tabular_classification	4	1.81201	HistGradientBoostingClassifier	fitted	-0.846154
TabularObjective_29442fec	TERMINATED	192.168.1.89:13259	tabular_classification	4	4.62255	BernoulliNB	fitted	-0.717949
TabularObjective_2a726e7a	TERMINATED	192.168.1.89:13262	tabular_classification	4	2.88022	Ltbsvm_svc	fitted	-0.632479
TabularObjective_2ebd7ab9	TERMINATED	192.168.1.89:13259	tabular_classification	4	2.93541	LogisticRegression	fitted	-0.846154
TabularObjective_2d4618b3	TERMINATED	192.168.1.89:13262	tabular_classification	4	11.6529	RandomForest	fitted	-0.846154
TabularObjective_2e985b00	TERMINATED	192.168.1.89:13259	tabular_classification	4	3.7879	GradientBoostingClassifier	fitted	-0.735043
TabularObjective_2f669952	TERMINATED	192.168.1.89:13259	tabular_classification	4	60.0609	QDA	fitted	-0.794872
TabularObjective_31b73aac	TERMINATED	192.168.1.89:13262	tabular_classification	4	5.8072	LogisticRegression	fitted	-0.777778
TabularObjective_35c6ead9	TERMINATED	192.168.1.89:13262	tabular_classification	4	1.53908	SGD	fitted	-0.837607
TabularObjective_39432256	TERMINATED	192.168.1.89:13262	tabular_classification	4	2.51557	PassiveAggressive	fitted	-0.709402

Fig. Report training process

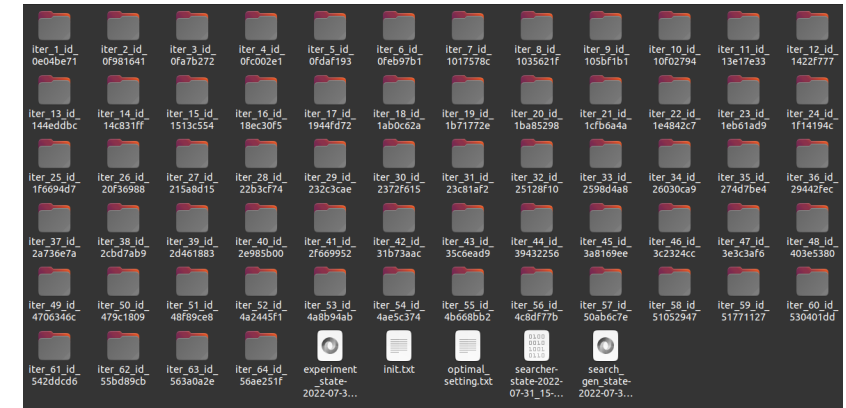


Fig. Store evaluation experiments

# 2.7. Model Selection & Hyperparameter Optimization

```
For pipeline 1:  
Optimal encoding method is: DataEncoding  
Optimal encoding hyperparameters: {}  
  
Optimal imputation method is: no_processing  
Optimal imputation hyperparameters: {}  
  
Optimal balancing method is: EditedNearestNeighbor  
Optimal balancing hyperparameters: {'imbalance_threshold': 0.9941343235473185, 'k': 1}  
  
Optimal scaling method is: MinMaxScale  
Optimal scaling hyperparameters: {}  
  
Optimal feature selection method is: no_processing  
Optimal feature selection hyperparameters: {}  
  
Optimal classification model is: RandomForest  
Optimal classification hyperparameters: {'bootstrap': True, 'criterion': 'entropy', 'max_depth': None, 'max_features': 0.5700333538264852, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 19, 'min_samples_split': 18, 'min_weight_fraction_leaf': 0.0}
```

Fig. Record optimal setting for checking

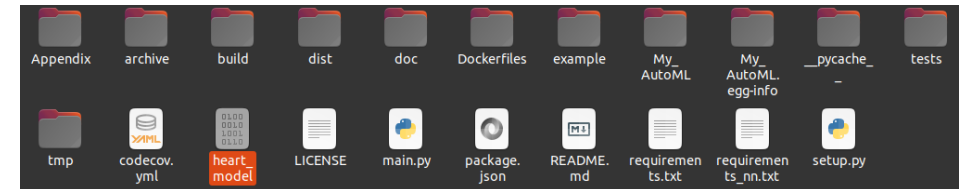


Fig. Store trained pipelines

## 2.7.1. Model Ensemble

To fight with data imbalance, model ensemble is another common solution, which is included in the pipeline.

Three commonly-used model ensembles are included:

1. **Stacking**

Models are trained parallelly on all train sets, but predictions are weighted as final prediction.

2. **Bagging**

Models are trained on subsets of train sets, and weighted predictions as final prediction.

3. **Boosting**

Models are trained on error of past models, and all predictions summed as final prediction.

# 3. Experiments

## 3.1. Heart Failure Prediction (Classification)

Command line job:

```
(AutoML) panyl@panyl-computer:~/Git Repo/My_AutoML$ python main.py --n_estimators 1 --data_folder example/example_data --train_data heart --response HeartDisease
```

Results (ignore all report process):

```
2022-07-31 15:43:49,978 INFO tune.py:747 -- Total run time: 183.64 seconds (183.17 seconds for the tuning loop).  
Evaluation:  
The accuracy of test data is: 0.8248
```

Results using 5 pipeline stacking:

```
2022-07-31 15:37:36,629 INFO tune.py:747 -- Total run time: 168.49 seconds (167.94 seconds for the tuning loop).  
Evaluation:  
The accuracy of test data is: 0.8467
```

# 3. Experiments

## 3.1. Heart Failure Prediction (Classification)

Kaggle Heart Failure Prediction dataset

<https://www.kaggle.com/fedesoriano/heart-failure-prediction>

```
[1] ✓ 2.2s Python
1 import My_AutoML
2 from My_AutoML import AutoTabular
3 from My_AutoML import load_data, train_test_split

[2] ✓ 0.5s Python
1 # load data
2 database = load_data(data_type=".csv").load("example/example_data", ["heart"])
3 database_names = [*database]
4 database_names

... ['heart']

[3] ✓ 0.9s Python
1 # examine data
2 database["heart"].head(5)
```

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	ExerciseAngina	Oldpeak	ST_Slope	HeartDisease
0	40	M	ATA	140	289	0	Normal	172	N	0.0	Up	0
1	49	F	NAP	160	180	0	Normal	156	N	1.0	Flat	1
2	37	M	ATA	130	283	0	ST	98	N	0.0	Up	0
3	48	F	ASY	138	214	0	Normal	108	Y	1.5	Flat	1
4	54	M	NAP	150	195	0	Normal	122	N	0.0	Up	0

Fig. Load data

# 3. Experiments

## 3.1. Heart Failure Prediction (Classification)

```
1 # train test split
2 X_train, X_test, y_train, y_test = train_test_split(database["heart"][features], database["heart"][[response]])
3 print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
```

✓ 0.1s

... (781, 11) (137, 11) (781, 1) (137, 1)

```
1 # fit AutoML model
2 mol = AutoTabular(progress_reporter = "JupyterNotebookReporter", seed = 42)
3 mol.fit(X_train, y_train)
```

✓ 4m 11.4s

... == Status ==  
Current time: 2022-07-31 16:59:26 (running for 00:04:03.43)  
Memory usage on this node: 6.4/11.6 GiB  
Using FIFO scheduling algorithm.  
Resources requested: 0/8 CPUs, 0/0 GPUs, 0.0/3.73 GiB heap, 0.0/1.86 GiB objects (0.0/1.0 accelerator\_type:G)  
Current best trial: 836a5170 with loss=-0.8290598290598291 and parameters={'task\_type': 'tabular\_classification'}  
Result logdir: /home/panyi/Git Repo/My\_AutoML/tmp/model  
Number of trials: 64/64 (64 TERMINATED)

Trial name	status	loc	task_type	iter	total time (s)	fitted_model	training_status	loss
TabularObjective_79f2edd0	TERMINATED	192.168.1.89:20681	tabular_classification	4	3.47034	PassiveAggressive	fitted	-0.786325
TabularObjective_7bc78be8	TERMINATED	192.168.1.89:20724	tabular_classification	4	16.5076	SGD	fitted	-0.794872
TabularObjective_7bc78be9	TERMINATED	192.168.1.89:20727	tabular_classification	4	3.89548	GAM_Classifier	fitted	-0.760684
TabularObjective_7bc78bea	TERMINATED	192.168.1.89:20733	tabular_classification	4	6.9044	GaussianNB	fitted	-0.786325
TabularObjective_7bc78beb	TERMINATED	192.168.1.89:20737	tabular_classification	4	2.74392	GradientBoostingClassifier	fitted	-0.538462
TabularObjective_7bc78bec	TERMINATED	192.168.1.89:20740	tabular_classification	4	5.11713	GaussianNB	fitted	-0.786325
TabularObjective_7bc78bed	TERMINATED	192.168.1.89:20769	tabular_classification	4	5.3532	LogisticRegression	fitted	-0.786325
TabularObjective_7cc3fcc	TERMINATED	192.168.1.89:20773	tabular_classification	4	5.96211	MLPClassifier	fitted	-0.769231
TabularObjective_7cc3fccb	TERMINATED	192.168.1.89:20681	tabular_classification	4	3.09094	GaussianNB	fitted	-0.683761

Fig. Train Test split & conduct experiment



# 3. Experiments

## 3.1. Heart Failure Prediction (Classification)

```
1 from sklearn.metrics import accuracy_score
2 y_pred = mol.predict(X_test)
3 print("Accuracy Score: {:.4f}".format(accuracy_score(y_test, y_pred)))
```

[8] ✓ 0.3s Python

... Accuracy Score: 0.8248

Fig. Evaluate Test Performance

# 3. Experiments

## 3.2. Insurance Premium Prediction (Regression)

Kaggle Insurance Premium Prediction dataset

<https://www.kaggle.com/noordeen/insurance-premium-prediction>

```
[1] ✓ 1.8s Python
1 import My AutoML
2 from My AutoML import AutoTabular
3 from My AutoML import load_data, train_test_split

[2] ✓ 0.7s Python
1 # load data
2 database = load_data(data_type=".csv").load("example/example_data", ["insurance"])
3 database_names = [*database]
4 database_names

... ['insurance']

[3] ✓ 0.9s Python
1 # examine data
2 database["insurance"].head(5)

...
  age  sex  bmi  children  smoker  region  expenses
0  19 female  27.9      0    yes southwest  16884.92
1  18  male  33.8      1    no  southeast   1725.55
2  28  male  33.0      3    no  southeast   4449.46
3  33  male  22.7      0    no northwest  21984.47
4  32  male  28.9      0    no northwest   3866.86
```

Fig. Load data

# 3. Experiments

## 3.2. Insurance Premium Prediction (Regression)

```
1 response = "expenses"
2 features = list(database["insurance"].columns)
3 features.remove(response)

(4) ✓ 0.1s Python

1 # train test split
2 X_train, X_test, y_train, y_test = train_test_split(database["insurance"][features], database["insurance"][response])
3 print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)

(3) ✓ 0.1s Python

... (1138, 6) (200, 6) (1138, 1) (200, 1)

1 # fit AutoML model
2 mol = AutoTabular(progress_reporter = "JupyterNotebookReporter", seed = 42)
3 mol.fit(X_train, y_train)

(4) ✓ 5m 38.9s Python

== Status ==
Current time: 2022-07-31 18:21:23 (running for 00:05:30.50)
Memory usage on this node: 8.1/11.6 GiB
Using FIFO scheduling algorithm.
Resources requested: 0/8 CPUs, 0/0 GPUs, 0.0/2.61 GiB heap, 0.0/1.31 GiB objects (0.0/1.0 accelerator type:G)
Current best trial: 3ebf7e8f with loss=11510832.441452352 and parameters={'task_type': 'tabular_regression'}
Result logdir: /home/panyi/Git Repo/My_AutoML/tmp/model
Number of trials: 64/64 (64 TERMINATED)

Trial name      status      loc      task_type  iter  total time (s)  fitted_model  training_status  loss
TabularObjective_b8ff4b8a  TERMINATED  192.168.1.89:31128  tabular_regression  4      11.4843      MLPRegressor  fitted  3.31961e+07
TabularObjective_bba07f08  TERMINATED  192.168.1.89:31174  tabular_regression  4      5.70969      GAM_Regressor  fitted  2.83755e+07
TabularObjective_bba07f09  TERMINATED  192.168.1.89:31177  tabular_regression  4      15.7685      RandomForest    fitted  3.81517e+07
TabularObjective_bba07f0a  TERMINATED  192.168.1.89:31179  tabular_regression  4      1.7417      SGD             fitted  4.86527e+07
TabularObjective_bf27d720  TERMINATED  192.168.1.89:31289  tabular_regression  4      264.882     HistGradientBoostingRegressor  fitted  1.91355e+08
TabularObjective_bf27d721  TERMINATED  192.168.1.89:31293  tabular_regression  4      7.10371     LightGBM_Regressor  fitted  6.2961e+08
TabularObjective_bf27d722  TERMINATED  192.168.1.89:31295  tabular_regression  4      142.014     GaussianProcess  fitted  1.40593e+08
TabularObjective_bf27d723  TERMINATED  192.168.1.89:31300  tabular_regression  4      2.35143     LinearRegression  fitted  3.30054e+07
TabularObjective_bf27d724  TERMINATED  192.168.1.89:31179  tabular_regression  4      34.7064     LibSVM_SVR       fitted  1.59965e+08
```

Fig. Train Test split & conduct experiment

# 3. Experiments

## 3.2. Insurance Premium Prediction (Regression)

```
1 from sklearn.metrics import mean_squared_error, r2_score
2 y_pred = mol.predict(X_test)
3 print("Mean Squared Error: {:.4f}".format(mean_squared_error(y_test, y_pred)))
4 print("R2 Score: {:.4f}".format(r2_score(y_test, y_pred)))
```

[8] ✓ 0.3s Python

... Mean Squared Error: 21671858.6434  
R2 Score: 0.8648

Fig. Evaluate Test Performance

# 4. Summary & Future

## Summary:

- (1) Provide a **workable** pipeline/framework for AutoML tasks.
- (2) **Performance** and **efficiency** of the pipeline for small datasets are at acceptable level.
- (3) For further improvement on accuracy, **increase the time budget** to allow more search & evaluations; or use current results as **baseline** to limit further search space.

## Future:

- (1) Modify the search space to allow faster training; Develop/Apply better search algorithm;
- (2) Find an applicable Neural Architecture Search (NAS) algorithm and hyperparameter optimization algorithm to expand allowed tasks.
- (3) AutoML usually is time-consuming, computational-expansive, thus, train on large datasets are not feasible, which limits its applications. Apply Few-shot, One-shot idea to improve efficiency.

# Reference

- [1] Mitchell, T. M., & Mitchell, T. M. (1997). *Machine learning* (Vol. 1, No. 9). New York: McGraw-hill.
- [2] <https://www.imeche.org/news/news-article/webinar-autonomous-vehicles---the-challenges-of-automated-driving>
- [3] Thornton, C., Hutter, F., Hoos, H. H., & Leyton-Brown, K. (2013, August). Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining* (pp. 847-855).
- [4] <https://auto.gluon.ai/stable/index.html>
- [5] <https://automl.github.io/auto-sklearn/master/>
- [6] <https://automl.github.io/Auto-PyTorch/development/>
- [7] <https://github.com/hibayesian/awesome-automl-papers/blob/master/resources/automl.png>
- [8] Musil, C. M., Warner, C. B., Yobas, P. K., & Jones, S. L. (2002). A comparison of imputation techniques for handling missing data. *Western journal of nursing research*, 24(7), 815-829.
- [9] Stekhoven, D. J., & Bühlmann, P. (2012). MissForest—non-parametric missing value imputation for mixed-type data. *Bioinformatics*, 28(1), 112-118.

# Reference

- [10] Azur, M. J., Stuart, E. A., Frangakis, C., & Leaf, P. J. (2011). Multiple imputation by chained equations: what is it and how does it work?. *International journal of methods in psychiatric research*, 20(1), 40-49.
- [11] Yoon, J., Jordon, J., & Schaar, M. (2018, July). Gain: Missing data imputation using generative adversarial nets. In *International conference on machine learning* (pp. 5689-5698). PMLR.
- [12] Batista, G. E., Prati, R. C., & Monard, M. C. (2004). A study of the behavior of several methods for balancing machine learning training data. *ACM SIGKDD explorations newsletter*, 6(1), 20-29.
- [13] Tomek, I. (1976). Two modifications of CNN.
- [14] <https://mlwhiz.com/blog/2020/01/28/imbal/>
- [15] Wilson, D. L. (1972). Asymptotic properties of nearest neighbor rules using edited data. *IEEE Transactions on Systems, Man, and Cybernetics*, (3), 408-421.
- [16] [https://www.researchgate.net/figure/Wilson-Editing-for-a-1-NN-Classifer\\_fig1\\_4133603](https://www.researchgate.net/figure/Wilson-Editing-for-a-1-NN-Classifer_fig1_4133603)

# Reference

- [17] Hart, P. (1968). The condensed nearest neighbor rule (corresp.). *IEEE transactions on information theory*, 14(3), 515-516.
- [18] Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. (2002). SMOTE: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16, 321-357.
- [19] <https://towardsdatascience.com/stop-using-smote-to-handle-all-your-imbalanced-data-34403399d3be>
- [20] Chandrashekar, G., & Sahin, F. (2014). A survey on feature selection methods. *Computers & Electrical Engineering*, 40(1), 16-28.
- [21] Somol, P., Pudil, P., Novovičová, J., & Paclík, P. (1999). Adaptive floating search methods in feature selection. *Pattern recognition letters*, 20(11-13), 1157-1163.
- [22] Tan, F., Fu, X., Zhang, Y., & Bourgeois, A. G. (2008). A genetic algorithm-based method for feature subset selection. *Soft Computing*, 12(2), 111-120.
- [23] Liu, Y., Tang, F., & Zeng, Z. (2014). Feature selection based on dependency margin. *IEEE Transactions on Cybernetics*, 45(6), 1209-1221.



# Reference

- [24] Peng, H., Long, F., & Ding, C. (2005). Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy. *IEEE Transactions on pattern analysis and machine intelligence*, 27(8), 1226-1238.
- [25] Li, Z. (2022). A Feature Selection Method Using Dynamic Dependency and Redundancy Analysis. *Arabian Journal for Science and Engineering*, 1-15.
- [26] Lall, S., Sinha, D., Ghosh, A., Sengupta, D., & Bandyopadhyay, S. (2021). Stable feature selection using copula based mutual information. *Pattern Recognition*, 112, 107697.
- [27] Lall, S., & Bandyopadhyay, S. (2019, September). An l1-Norm regularized copula based feature selection. In *Proceedings of the 2019 3rd International Symposium on Computer Science and Intelligent Control* (pp. 1-6).
- [28] Hastie, T., Rosset, S., Zhu, J. & Zou, H. Multi-class adaboost. *Statistics and its Interface* **2**, 349–360 (2009).
- [29] Safavian, S. R. & Landgrebe, D. A survey of decision tree classifier methodology. *IEEE transactions on systems, man, and cybernetics* **21**, 660–674 (1991).

# Reference

- [30] Fan, R.-E., Chang, K.-W., Hsieh, C.-J., Wang, X.-R. & Lin, C.-J. Liblinear: A library for large linear classification. *the Journal of machine Learning research* **9**, 1871–1874 (2008).
- [31] Chang, C.-C. & Lin, C.-J. Libsvm: a library for support vector machines. *ACM transactions on intelligent systems and technology (TIST)* **2**, 1–27 (2011).
- [32] <https://medium.com/@viveksalunkhe80/support-vector-machine-svm-88f360ff5f38>
- [33] Gardner, M. W. & Dorling, S. Artificial neural networks (the multilayer perceptron)—a review of applications in the atmospheric sciences. *Atmospheric environment* **32**, 2627–2636 (1998).
- [34] Wilhelm, F. The idea behind automatic relevance determination and bayesian interpolation. <https://www.youtube.com/watch?v=2gT-Q0NZzoE> (2016).
- [35] Wang, J. An intuitive tutorial to gaussian processes regression. *arXiv preprint arXiv:2009.10862* (2020).
- [36] Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., ... & Liu, T. Y. (2017). Lightgbm: A highly efficient gradient boosting decision tree. *Advances in neural information processing systems*, 30.

# Reference

- [37] Chen, T., & Guestrin, C. (2016, August). Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining* (pp. 785-794).
- [38] Servén D., Brummitt C. (2018). pyGAM: Generalized Additive Models in Python. Zenodo. [DOI: 10.5281/zenodo.1208723](https://doi.org/10.5281/zenodo.1208723)
- [39] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *the Journal of machine Learning research*, 12, 2825-2830.
- [40] Feurer, M., Klein, A., Eggenberger, K., Springenberg, J., Blum, M., & Hutter, F. (2015). Efficient and robust automated machine learning. *Advances in neural information processing systems*, 28.
- [41] <https://www.freecodecamp.org/news/how-to-use-the-tree-based-algorithm-for-machine-learning/>
- [42] <https://online.stat.psu.edu/stat508/book/export/html/696>
- [43] <https://towardsdatascience.com/from-linear-regression-to-ridge-regression-the-lasso-and-the-elastic-net-4eaecaf5f7e6>
- [44] <https://towardsdatascience.com/understanding-adaboost-2f94f22d5bfe>
- [45] <https://towardsdatascience.com/knn-visualization-in-just-13-lines-of-code-32820d72c6b6>

# Reference

- [46] <https://suniljangirblog.wordpress.com/2018/12/13/variants-of-gradient-descent/>
- [47] Liaw, R., Liang, E., Nishihara, R., Moritz, P., Gonzalez, J. E., & Stoica, I. (2018). Tune: A research platform for distributed model selection and training. *arXiv preprint arXiv:1807.05118*.
- [48] Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., ... & Zheng, X. (2016). {TensorFlow}: a system for {Large-Scale} machine learning. In *12th USENIX symposium on operating systems design and implementation (OSDI 16)* (pp. 265-283).
- [49] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., ... & Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32.
- [50] Akiba, T., Sano, S., Yanase, T., Ohta, T., & Koyama, M. (2019, July). Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining* (pp. 2623-2631).
- [51] Bergstra, J., Yamins, D., & Cox, D. D. (2013, June). Hyperopt: A python library for optimizing the hyperparameters of machine learning algorithms. In *Proceedings of the 12th Python in science conference* (Vol. 13, p. 20).

All code files, report and presentations are available at: [https://github.com/PanyiDong/My\\_AutoML.git](https://github.com/PanyiDong/My_AutoML.git)

PanyiDong / My\_AutoML (Public)

<> Code Issues Pull requests Actions Projects Wiki Security Insights Settings

master 2 branches 3 tags Go to file Add file + Code

PanyiDong correct error max\_depth\_factor mixed max\_depth ✓ 9945a1f · 25 days ago 505 commits

📁 .github	update build names	3 months ago
📁 Appendix	update test methods	4 months ago
📁 Dockerfiles	update	3 months ago
📁 My_AutoML	correct error max_depth_factor mixed max_depth	25 days ago
📁 archive	complete re-constructure folders	4 months ago
📁 doc	change directory of doc	3 months ago
📁 example	correct typos and change class names	5 months ago
📁 tests	correct test errors	3 months ago
📄 .gitattributes	test gitattributes	4 months ago
📄 .gitignore	update README, setup.py and workflow installation	3 months ago
📄 LICENSE	Create LICENSE	4 months ago
📄 README.md	add model ensemble	3 months ago
📄 codecov.yml	put wrap classes for every imported models	4 months ago
📄 main.py	add model ensemble	3 months ago
📄 package.json	update tests, new version	3 months ago
📄 requirements.txt	correct tests	4 months ago
📄 requirements_nn.txt	correct tests	4 months ago
📄 setup.py	update	3 months ago

📄 README.md

## Project for Auto Machine Learning (AutoML)

Linux passing Windows passing release v0.2.1 codecov 92% build-nn passing

IRisk Lab Project, UIUC, Fall 2021

Now a personally-maintained project

The project aims to create a AutoML package with special focus on insurance data (with some imbalance in nature). The pipeline is now workable with encoding, imputation, balancing, scaling, feature selection, models (regression, classification models) as pipeline components and model selection/hyperparameter optimization (HPO) process as it's core of connection among all components for tuning.

About

AutoML pipeline project.

Readme

MIT license

1 star

1 watching

0 forks

Releases 3

v0.2.1 Full Test/Coverage wor... Latest  
on Apr 30

+ 2 releases

Packages

No packages published

Publish your first package

Languages

Python 96.3% Jupyter Notebook 3.3% Other 0.4%

Tutorials on installment, usage are all available at the page.

# Thanks!

# Questions