Automated Machine Learning (AutoML)

Panyi Dong

1 Introduction

Machine Learning (ML), the approach of adopting statistical methods to learn the structure of datasets using computer algorithms and fulfilling certain tasks [1], has grown more and more popular during the past few decades with a wide range of both industrial and academic applications, like predicting future stock prices for financial decision making, classifying the new articles to categories so that people can easily find out useful information in desired topics, recommending content for your potential interest on social media, etc. With massive amount of resources devoted, the ML area has gained explosive developments, and enormous amount of ML models has been developed, ranging from simplest linear regression, random forest, to complicated neural networks with millions or even billions of parameters tuned (175 billion parameters for GPT-3 network). As a result, the problem of which model to select among all possible choices and how to efficiently tune the selected model to reach optimal performance rises. Traditionally, human expertise and experience are necessary and long period of train/validation time is required for manual optimization, which makes ML difficult to implement for non-experts or for large size networks. The topic of Automated Machine Learning (AutoML) aims to solve the problem. The general idea of AutoML is to create a robust pipeline for ML models, which aims to deal with all ML tasks regardless of the input datasets automatically with less human intervention as possible. By thus, AutoML allows ML more accessible to general audience with no expertise or previous experience [2] or create a baseline for further optimization.

The method of AutoML then raises other new problems including how to define the model space we search (create a pool of ML models), how to search for the best models, how to optimize the models, how to improve the efficiency of entire process while still maintaining acceptable performance, etc. This paper introduces a AutoML pipeline with special concentration on Insurance datasets where imbalanced nature can sometimes negatively impact general AutoML pipeline performance (detailed explanation in section 2.3 Data Balancing). The AutoML pipeline consists of several preprocessing steps and a model selection, hyperparameter optimization combined step based on Hyperopt to select optimal model and hyperparameter setting. Testing on several real-world datasets demonstrates that the pipeline can reach overall good performance and can serve as a baseline for further optimization.

The report comes in the following order: Chapter 2 introduces the preprocessing methods available in the pipeline (2.1 for Data Encoding, 2.2 for Data Imputation, 2.3 for Data Balancing, 2.4 for Data Scaling, 2.5 for Data Feature Selection); Chapter 3 introduces the classification/regression models available in the pipeline and model selection/hyperparameter optimization process employed in the pipeline; Chapter 4 demonstrates tests of the AutoML pipeline on two datasets; Chapter 5 summaries the report and introduces some future improvements.

For now, the AutoML pipeline can only deal with tabular datasets (with input of pandas Dataframe) and supervised learning tasks (classification and regression).

2 Preprocessing

2.1 Data Encoding

For some of the datasets, there are categorical string type data information (like ”M”/”F” for gender information, ”Yes”/”No” information, or multiple categorical variables). These type of features are not easy to deal with in the pipeline or can not be processed due to the intrinsic nature of pipeline. Thus, it's crucial to convert these features to numerical ones.

In the pipeline, the categorical features will be converted to numerical ones by the unique values ("M" to 0, "F" to 1; or "Green" to 0, "Red" to 1, "Yellow" to 2, ...).

2.2 Data Imputation

One common problem raised from real-life datasets is that due to reasons like lacking of proper storage, people deliberately/unconsciously fails to provide the information, etc., we sometimes can only get datasets with missing values. However, most of ML models are not designed or capable of dealing with missing values. Thus, imputation is a crucial step to fill these missing values properly. An appropriate imputation method can generate most relevant values while poor choice of imputation methods can lead to bad performance.

In the pipeline, if there's no missing values found, the imputation step will be skipped and not defined in the evaluation process.

Since in real-life insurance datasets, missing values can sometimes contain certain information (people deliberately provide missing past medical history to become qualified to enroll in health insurance). Simply imputing missing values with mean/zero fails to retain the information in data structure, thus, is not an ideal imputation method in most circumstances. Sophisticated imputation methods that can retain the information in data structure are required. The imputation methods available in the AutoML are:

2.2.1 Simple Imputation

Simple Imputation is the simplest and one of the most common imputation method where the goal is to fill the missing values with mean/zero/median of the features. The method is most efficient since there's no need to consider the correlation between features and observations with missing values, but it's the exact shortage of using this method. The filled values should be generated based on the data structure and represent the observations' information, thus imputation using Simple Imputation can sometimes lead to negative impact on the model.

2.2.2 Joint Imputation

Joint Imputation consider the entire dataset as a joint distribution (like multivariate normal distribution). It will calculate the distribution parameters (like mean vector μ and covariance matrix Σ in multivariate normal distribution) using the observations with no missing values and impute the missing values.

For multivariate normal kernel, take *obs* as no missing observations, and *mis* as observations with missing values, for

$$\vec{x} = \begin{pmatrix} \vec{x}_{mis} \\ \vec{x}_{obs} \end{pmatrix}$$

with

$$\vec{\mu} = \begin{pmatrix} \vec{\mu}_{mis} \\ \vec{\mu}_{obs} \end{pmatrix}, \vec{\Sigma} = \begin{pmatrix} \vec{\Sigma}_{mis,mis} & \vec{\Sigma}_{mis,obs} \\ \vec{\Sigma}_{obs,mis} & \vec{\Sigma}_{obs,obs} \end{pmatrix}$$

then, the conditional distribution of

$$a = \vec{x}_{mis} | \vec{x}_{obs}$$

is $N(\bar{\mu}, \bar{\Sigma})$, where

$$\bar{\mu} = \vec{\mu}_{mis} + \vec{\Sigma}^{-1}_{obs,obs}(a - \vec{\mu}_{obs}),$$

$$\bar{\Sigma} = \vec{\Sigma}_{mis,mis} - \vec{\Sigma}_{mis,obs}\vec{\Sigma}^{-1}_{obs,obs}\vec{\Sigma}_{obs,mis}.$$

2.2.3 Expectation Maximization

Expectation Maximization (EM) is an iterative method to find maximum likelihood estimates of parameters. For EM imputation, the algorithm iteratively perform E step (create a function for expectation) and M step (find the parameters maximize the likelihood) to fill the missing values to maximize the likelihood.

2.2.4 KNN Imputation [3]

KNN Imputation uses the K Nearest Neighbors methods to find most appropriate values to fill missing positions. KNN method finds closest data points and fill with the corresponding values. Since the spatial positions (usually calculated by similarity matrix) of data points usually represent observation information (response), it's reasonable to use KNN to fill the missing values.

In practice, the algorithm fills the missing values using Simple Imputation and take multiple rounds of imputation using KNN for the missing values positions. And since K is a crucial hyperparameter for KNN method, the algorithm will run cross validation to select best K for imputation.

2.2.5 Miss Forest Imputation [3]

Miss Forest Imputation uses Random Forest as prediction phase. The algorithm uses non-missing observations as train data and others as test data. The algorithm will train a Random Forest model for the train data and predict on test data (the prediction will be imputation data). Since one round of Random Forest imputation is usually not stable, the algorithm will calculate the difference between pre-imputed data and imputed data, and only when the difference converges will the algorithm stops.

2.2.6 MICE [4]

Multiple Imputation by Chained Equations (MICE) will run Linear Regression/Logistic Regression/Lasso on the non-missing values and use the prediction on missing values as imputed data.

In the process, the algorithm first impute using mean values (same as Simple Imputation) but records the missing places. Then, set recorded missing places for one feature are set to missing again and train a regression model on non-missing values, impute using the prediction. Perform the procedure for every feature containing missing places, which is considered as one round of imputation. Then, iteratively repeat above procedures for several rounds until the alterations on the datasets converge.

2.2.7 GAIN [5]

Generative Adversarial Imputation Nets (GAIN) is a Generative Adversarial Network (GAN) based method. The network will train a Generator (G) and Discriminator (D) to compete and impute the missing values.

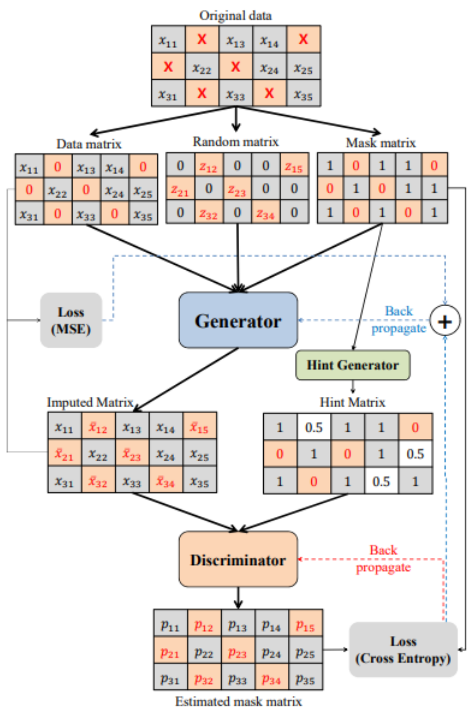


Figure 1: Architecture of GAIN [5]

As shown in Fig1, the network will compute a Generator using Data matrix (with missing values), Random matrix (random imputation at missing places) and Mask matrix (indicates where are the missing places) to form Imputed matrix and combining Imputed matrix and Hint matrix (generated from Mask matrix) to create a Discriminator. The Generate and Discriminator compete with each other (represented by weights on matrix) to get ultimate Imputed matrix.

2.3 Data Balancing [6]

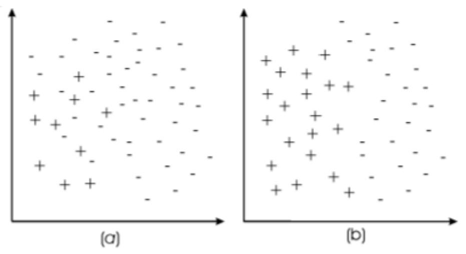


Figure 2: Unbalanced Data Points (a); Balanced Data Points (b) [6]

Insurance datasets usually have imbalanced nature, where one class (usually 0) can take majority of the observations. This is reasonable since for insurance policies, most of the policyholders do not report any claims in the covered period, or at least only file a small claim amount (large claim size are extreme cases and quite uncommon). However, the claims are the core of risk quantifying since they cause losses for insurance companies. The imbalanced datasets are not ideal for most of ML models since for these models, all observations are equally weighted and the minority class (losses we focus on) is not weighted enough to contribute to final decision boundary. Thus, we have to balance the datasets. As shown in Fig2, (a) shows an unbalanced dataset where only few + data points exists, while (b) demonstrates a well-balanced datasets. The goal of this step is to raise the percentage of minority class to a level where it's comparable to majority class.

In the procedure, the balance limit is a tunable hyperparameter for all balancing methods. Although the pipeline will always contain a balancing step, if the balance limit is above data majority percentage, there will be no actual balancing step.

Common solutions are under-sampling and over-sampling where the idea is to synthetically remove majority class observations or generate minority class observations (under-sampling and over-sampling respectively). In the AutoML pipeline, the balancing methods I used are (all belong to over-sampling/under-sampling):

2.3.1 Simple Random Over-Sampling

Simple Random Over-Sampling takes the simple idea that randomly select observations in minority class and duplicate the selected observations. By thus, the percentage of minority class increases and the effect of imbalance reduces. However, since the newly-duplicated observations are the same as those in original datasets, the accuracy of decision boundary can not be improved drastically.

2.3.2 Simple Random Under-Sampling

For Simple Random Under-Sampling, the idea is to randomly select observations in majority class and remove these selected observations. The procedure will decrease the percentage of majority class. The problem with this method is that some observations in majority class are crucial for defining decision boundary while others are not that important. However, random removal of majority observations can remove those which are significant for decision boundary and worsen the model's performance.

2.3.3 Tomek Link [7]

Tomek Link is method of Under-Sampling. The idea of Tomek Link is to remove noise or border significant samples in majority class. A Tomek Link is defined as the pairs of observations E_i, E_j that belong to different classes (majority and minority class in this case) and for all other observations E_l, $d(E_i, E_j) < d(E_i, E_l)$ and $d(E_i, E_j) < d(E_j, E_l)$ (E_i and E_j being closest pair of observations). Since some of the noise observations belong to majority class while also in the decision boundary of minority class, Tomek Link intends to remove those noise observations.

In practice, the algorithm randomly selects observations in minority class and finds the nearest neighbor of the selected observations. If the nearest neighbor belongs to majority class, it will be removed from the dataset. For Tomek Link, since only majority class observations are removed in the procedure, it's a Under-Sampling process.

2.3.4 Edited Nearest Neighbor [8]

Edited Nearest Neighbor (ENN) is a under-sampling method adopted from kNN. In the procedure, the algorithm randomly select one observation and k nearest neighbors. If the random observation belongs to majority class, and the prediction using nearest neighbors disagrees with the random observation, then remove the random observation; if the random observation belongs to minority class while the prediction using nearest neighbors disagree with the observation, then those nearest neighbors will be removed.

The k of nearest neighbors is a hyperparameter and crucial for the performance of ENN. It can be tuned in the pipeline.

2.3.5 Condensed Nearest Neighbor [9]

Condensed Nearest Neighbor (CNN) is a Under-Sampling method to find a consistent subset samples of the entire dataset. The algorithm intends to select the consistent subset $\hat{E} \subseteq E$ where the $\hat{E}$ can predict all observations in E correctly with 1-nearest neighbor.

In the procedure, create a initial subset $\hat{E}$ by combining all observations in minority class and one random observation from majority class. Then predict the all observations in E by $\hat{E}$ using 1-nearest neighbor. After that, put all mis-predicted observations into the $\hat{E}$ as new $\hat{E}$. The idea of CNN is to remove those observations distant from decision boundary since those observations are considered less relevant with the learning process.

2.3.6 One Sided Selection

One Sided Selection (OSS) a Under-Sampling method with a combination of Tomek Link followed by CNN. Tomek Link is used to remove noise and decision boundary significant observations and CNN to remove observations distant from decision boundary. The two-step Under-Sampling method takes advantages of both removing boundary significant noise and those distant from decision boundary. The two-step method can achieve better performance compared to previous one-step methods.

2.3.7 CNN-Tomek Link

Compared to OSS, CNN-Tomek Link switches the two steps and run in the order of CNN and Tomek Link instead. The advantage of CNN-Tomek Link is that since Tomek Link requires a calculation of distances between all observations, and computational expensive, CNN first for reduced dataset size can reduce the computations required for Tomek Link and increase the efficiency.

2.3.8 Smote [10]

Synthetic Minority Over-sampling Technique (Smote) is a Over-Sampling method for synthetic minority observations generation.

Compared with Simple Random Over-Sampling, where the generated synthetic minority class observations are exact same from original datasets, Smote generates new minority class observations by interpolating between several minority class observations.

In process, the algorithm randomly selects two minority class observations and use the midpoint or other interpolating point along the two observation vector as new synthetic minority class observation. By thus, the synthetic minority observations will be brand-new and never-seen ones. Since the synthetic minority observations are generated using interpolation, most of the new observations are within minority decision boundary and beneficial for defining decision boundary.

2.3.9 Smote-Tomek Link

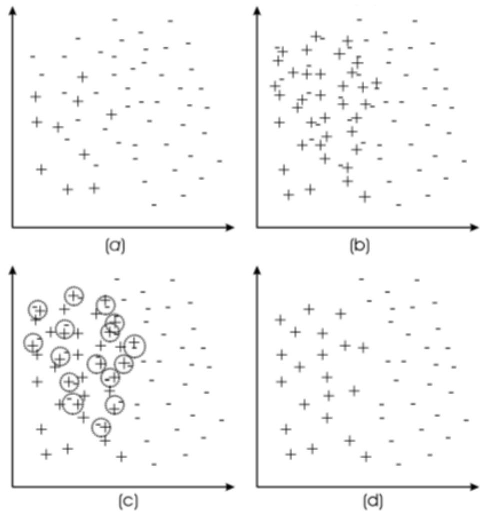


Figure 3: Original Unbalanced Data Points (a); Over-Sampling Data Points (b); Tomek Link Identification (c); Removal of Noise and Boderline Observations (d) [6]

Smote-Tomek Link is a compound balancing method by first Over-Sampling using Smote and then Under-Sampling using Tomek-Link. Smote for Over-Sampling can balance the extreme classes but fail to solve the problem of data skewness. For some of the datasets, the minority class observations can be invading into majority class space and vice versa (as shown in Fig3(b)), and Somte using interpolation can sometimes further worsen the circumstances. However, the benefit of Tomek Link to remove noise and boundary significant observations suits the need of balance data skewness (as shown in Fig3(c), (d)). Thus, applying Tomek Link after Smote can create a better-defined class clusters.

2.3.10 Smote-ENN

The idea of Smote-ENN is similar to Smote-Tomek Link, which is to provide a better-defined class space for learning after Over-Sampling using Smote. However, since ENN removes more observations compared to Tomek Link, Smote-ENN can provide more in depth data cleaning.

2.4 Data Scaling

Since the range of some features in the datasets varies (the house price can be tens of thousands or can be tens of millions), some models that require convergence or iterations can take tens of thousands of rounds of computation. Thus, in some cases, scaling on the datasets are necessary to improve efficiency or sometimes increase accuracy. The scaling methods used in this AutoML pipeline are:

2.4.1 Standardize

Standardization is one of the most common scaling methods. The features are standardized using $(\vec{x} - \bar{x})/\sigma_{\vec{x}}$.

2.4.2 Normalize

Normalization scales the features by $\vec{x}/\vec{x}_{max}$ so that features will be in unit scale.

2.4.3 Robust Scaling

Robust Scaling scales the features by two feature quantiles (25 percent quantile and 75 percentile quantile by default).

2.4.4 Min-Max Scaling

Compared with Normalization, Min-Max Scaling scales the features by $\vec{x}/(\vec{x}_{max} - \vec{x}_{min})$.

2.4.5 Winsorization

Winsorization removes extreme observations (default by 95 percent) so outliers will not negatively affect the models.

2.5 Data Feature Selection

For datasets, there are usually numerous features, however, not all features are relevant to the response (Age and Past Medical History are relevant to whether the person will die from heart attack, but the person's birthing month is much less relevant). So, this step of pipeline intends to select most relevant features for the model. The reduction of numbers of features can reduce the computation resources required while still maintains the model's performance.

For the AutoML pipeline, available feature selection methods include: Extra Tree for Feature Selection, Support Vector Machine for Feature Selection, Percentile for Feature Selection, etc. Furthermore, some complicated feature selection methods that deserve explanations are:

2.5.1 Feature Filter [11]

Feature Filter is a method to select relevant features by scoring features and retain the highest few.

One criterion is to score the features by calculating Pearson correlation coefficient, correlation between i-th feature and response is:

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

the higher the correlation, the more relevant the feature with response and the algorithm will select only features with highest correlation.

Another possible criterion is Mutual Information (MI) defined using Shannon entropy. The uncertainty (information content) in response Y is:

$$H(Y) = -\sum_y p(y)\log(p(y))$$

And conditional entropy for feature X is:

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

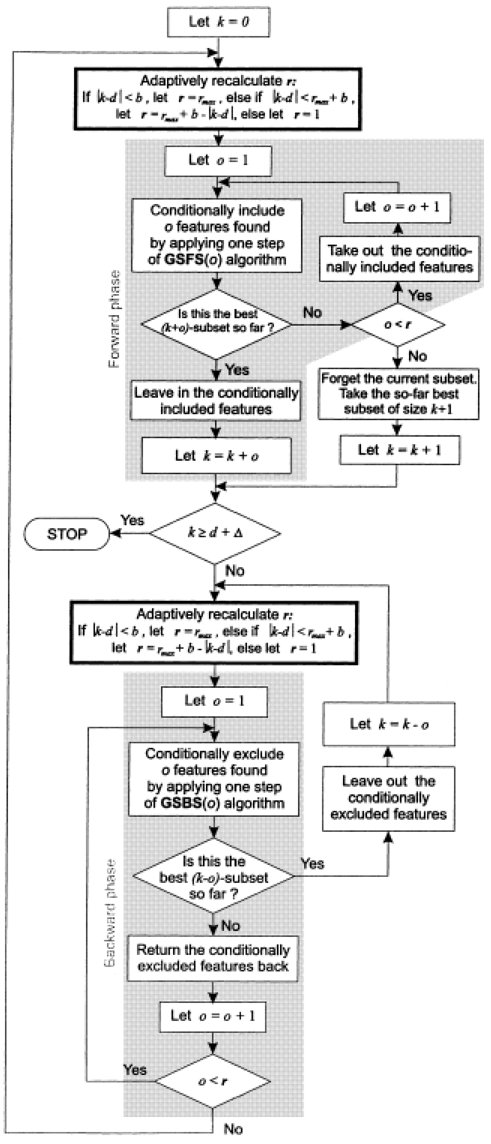
The MI between feature X and response Y is then defined as:

$$I(X, Y) = H(Y) - H(Y|X)$$

Once MI for all features are calculated, the algorithm only retains the features with highest features.

2.5.2 ASFFS [12]

Adaptive Sequential Forward Floating Search (ASFFS) is one of a sequential feature selection methods along with Sequential Backward Selection (SBS), Sequential Forward Selection (SFS), Adaptive Floating Search (AFS) where the core of sequential selection is to recursively add/remove features that can improve the model's performance. However, the adding/removal of features usually miss the correlation between features, where sometimes, certain combination of features can achieve better performance. The ASFFS algorithm, on the other hand, recursively add/remove features with adaptive number of features at a time. Since the adaptive number includes the correlation between features, ASFFS algorithm can achieve better performance.



Let $k = 0$
Adaptively recalculate r: If $|k-d| < b$, let $r = r_{max}$, else if $|k-d| < r_{max} + b$, let $r = r_{max} + b - |k-d|$, else let $r = 1$
Let $o = 1$
Conditionally include o features found by applying one step of **GSFS**(o) algorithm
Let $o = o + 1$
Take out the conditionally included features
Forward phase
Is this the best $(k+o)$-subset so far ?
No
Yes
$o < r$
Yes
No
Leave in the conditionally included features
Forget the current subset. Take the so-far best subset of size $k+1$
Let $k = k + o$
Let $k = k + 1$
STOP
Yes
$k \geq d + \Delta$
No
Adaptively recalculate r: If $|k-d| < b$, let $r = r_{max}$, else if $|k-d| < r_{max} + b$, let $r = r_{max} + b - |k-d|$, else let $r = 1$
Let $o = 1$
Let $k = k - o$
Conditionally exclude o features found by applying one step of **GSBS**(o) algorithm
Leave out the conditionally excluded features
Backward phase
Is this the best $(k-o)$-subset so far ?
Yes
No
Return the conditionally excluded features back
Let $o = o + 1$
Yes
$o < r$
No

Figure 4: Workflow of ASFFS [12]

The ASFFS algorithm will iteratively execute a Forward phase to add features to selection rule and a Backward phase to remove features in selection rule as shown in Fig4 until predefined resulting feature number is reached.

2.5.3 Genetic Algorithm [13]

Genetic Algorithm (GA) is a popular, effective algorithm in Machine Learning field and is inspired by natural evolution/mutation process. In the AutoML pipeline, GA is adopted for feature selection.

During the procedure, first, the algorithm uses some predefined features selection methods (Entropy for feature scoring, t-statistics for feature scoring and SVM-RFE/Support Vector Machine based Recursive Feature Elimination) or random selection to generate a pool of binary feature selection rules. The rules only consists of 1 (being selected) and 0 (not selected).

During the GA steps, recursively executes three steps process:

(1) Selection: apply Roulette Wheel Selection, rank every individual feature selection rules by predefined fitness function and assign possibilities to every rules, then randomly select selection rules according to the assigned possibilities;

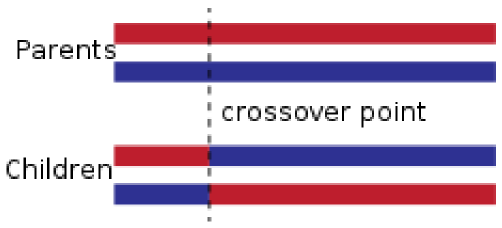


Figure 5: Single Point Crossover [14]

(2) Crossover: apply Single-point Crossover operation (as shown in Fig5), create child selection rules from parents rules which are selected in the first step by exchanging parts of the rules. The step tries to mimic the natural DNA activities.

(3) Mutation: Mutate n bits (here one bit indicates whether to select one feature represented by 0/1) in the child rules generated above by certain possibilities. Mutate in this circumstance means to flip 1 to 0 or 0 to 1. The step tries to mimic the mutation in natural biology behaviors. The mutation possibilities are small since mutation do not happen

heavily in multi-cell organisms and mutation do not guarantee better feature selection rules but only possibility of resulting better rules.

With generations of GA steps, the algorithm will select the best-performing rules, or if there's no significant improvement in the top N selection rules, the GA algorithm will perform early stop to increase the efficiency.

3 Model Selection and Hyperparameter Optimization

3.1 ML Models

After dataset preprocessed, the pipeline need to perform a classification /regression model. Due to time limitation, the AutoML pipeline consists of few traditional ML models provided by auto-sklearn [15].

Classification models :

1. Adaboost Classifier: Adaboost combines many weak tree classifiers to approximate Bayes classifier. The algorithm starts by creating a classification tree, then increases the weight of misclassified data points, and train a new classification tree using these new weights. Iteratively builds new classification trees and updates the weights. Ultimately, combine these trained classification trees, i.e. 500 trees with different weights, with score for each of them, to predict test data. Adaboost is beneficial for two-class classification, but not very successful for multi-class classification (caused by requirement for boosting). SAMME(Stagewise Additive Modeling using a Multi-class Exponential function) can improve the performance of AdaBoost by adding log(K - 1) to put more weight on misclassified data. [16]

2. Bernoulli Naive Beyas: Using Bayes Theorem to calculate the probability of belonging to certain class, but with kernel of Bernoulli/multivariate Bernoulli.

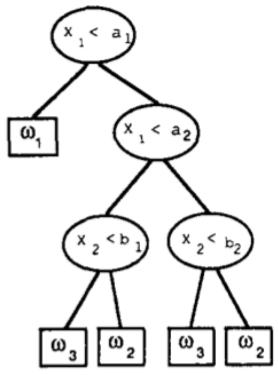


Figure 6: Decision Tree in tree format [17]

3. Decision Tree: Create a tree (as shown in Fig6) and assign classes to leaf nodes for classification.

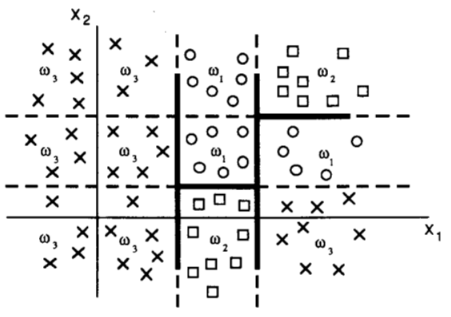


Figure 7: Decision Tree in Data Division [17]

The essential of tree type models is to divide the data points into different categories spatially (as shown in Fig7, a two-dimension decision tree division).

4. Extra Trees Classifier: Use multiple randomized decision trees based on subsamples of the dataset, and average the trees to improve prediction efficiency.

5. Gaussian Naive Beyas: Using Bayes Theorem to calculate the probability of belonging to certain class, but with kernel of Normal/multivariate Normal.

6. Gradient Boosting Classifier: Build a decision tree and iteratively build trees based on the residuals. The final classification model is an ensemble of these decision trees.

7. K Nearest Neighbors (kNN) Classifier: Classify the data with weighted average of k nearest neighbors using distance matrix or similarity matrix.

8. Linear Discriminant Analysis (LDA): Assume two Gaussian distributed classes share same covariance matrix but with different mean vectors, the decision boundary of two classes by:

$$\log(\pi_1)+(\vec{x}-\vec{\mu}_1)^T\Sigma^{-1}(\vec{x}-\vec{\mu}_1) = \log(\pi_2)+(\vec{x}-\vec{\mu}_2)^T\Sigma^{-1}(\vec{x}-\vec{\mu}_2)$$

   The decision boundary of LDA is a linear line.

9. LibLinear_SVC: The Support Vector Machine solves the optimal decision boundary by finding $\mathbf{w}$ that $\min_{\mathbf{w}} \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_i \xi(\mathbf{w};\mathbf{x}_i,y_i)$, where $\xi(\mathbf{w};\mathbf{x}_i,y_i)$ refers to loss function, which is penalized by C. The method is based on library *LibLinear* [18]

10. LibSVM_SVC: The same Support Vector Machine idea but with support of library *LibSVM.* [19]

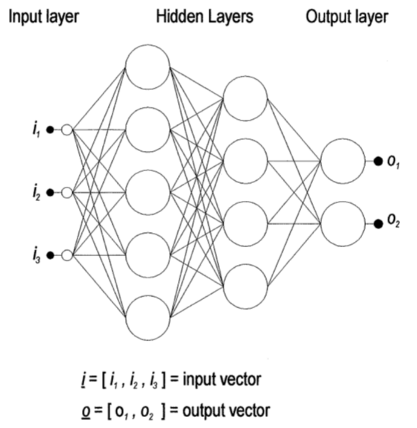

Input layer
Hidden Layers
Output layer
i_1
i_2
i_3
o_1
o_2
$\underline{i} = [i_1, i_2, i_3]$ = input vector
$\underline{o} = [o_1, o_2]$ = output vector

Figure 8: Illustration of MLP [20]

11. MultiLayer Perceptron (MLP) Classifier: MLP is a simple neural network architecture that belongs to Feedforward Neural Network structure, where the layers are connected with each other and consists of multiple hidden layers and non-linear activation functions (as shown in Fig8, a two-layer MLP model).

12. Multinomial Naive Beyas: Using Bayes Theorem to calculate the probability of belonging to certain class, but with kernel of Multinomial.

13. Passive Aggressive: Passive Aggressive is one of the online learning algorithms. The algorithm sequentially feed the model with small batch of data, update the classifier, until we get a good model. During the process, if the classification is correct, the model will be retained (passive); however, if the classification is wrong, the classifier will be updated to adjust to misclassified data points (aggressive).

14. Quadratic Discriminant Analysis (QDA): Assume two Gaussian distributed classes share different covariance matrix and different mean vectors, the decision boundary of two classes by:

$$\log(\pi_1)+(\vec{x}-\vec{\mu}_1)^T\Sigma_1^{-1}(\vec{x}-\vec{\mu}_1)-\log(\Sigma_1) = \log(\pi_2)+(\vec{x}-\vec{\mu}_2)^T\Sigma_2^{-1}(\vec{x}-\vec{\mu}_2)-\log(\Sigma_2)$$

   The decision boundary of QDA is a quadratic line.

15. Random Forest: Random Forest is a way of averaging multiple deep decision trees and improve the model performance. Random Forest randomly select subset of features at each split.

16. Stochastic Gradient Descent (SGD): Use random observation to calculate gradient and achieve the goal of minimizing loss function.

Regression models (some of the models applies the same idea in the classification models but with some modifications for regression tasks, so no explanation provided):

1. Adaboost Regressor

2. ARD Regressor: In autosklearn, ARD Regression uses Bayes ARD regression, where the weights of regression model assumed to be Gaussian distributed. The algorithm uses Bayes posterior distribution to build predictive distribution.

   Compared with Bayesian Ridge Regression, each weight in ARD has individual variance, i.e. $(\mathbf{w}|\lambda) \sim N(0,\Lambda^{-1})$, where $\Lambda = diag(\lambda_1,\lambda_2,...,\lambda_p)$ for p features. And optimal problem can be rewritten as $\min_{\alpha,\lambda} \alpha||\phi\mathbf{w}-\mathbf{t}||_2^2+\mathbf{w}^T\Lambda\mathbf{w}$. This allows features selection, since weight w of certain features can be close to 0 and we can prune these features. Moreover, compared with Cross Validation, if features are comparable to observations, Cross Validation can causes problems (splitted train set n ¡ p), but Bayesian Ridge Regression can avoid this situation. [21]

3. Decision Tree
4. Extra Trees Regressor
5. Gaussian Process: For Gaussian Process, means of distributions are typical set to 0 or sample mean, but the covariance matrix can be of infinite choices as kernel functions. Infinite dimension of kernel functions can be used to fit any data points. Thus Multivariate Normal Distribution is used as prior function in Bayesian approach. Build the probability distribution over all admissible functions that fits the train data. The predictive function can be constructed correspondingly. [22]
6. Gradient Boosting
7. K Nearest Neighbors Regressor
8. LibLinear_SVR [18]
9. LibSVM_SVR [19]
10. MultiLayer Perceptron (MLP) Regressor
11. Random Forest
12. Stochastic Gradient Descent (SGD)

3.2 Model Selection and Hyperparameters Optimization using Hyperopt

With the help of Hyperopt, the model selection and hyperparameter optimization can be easily implemented. In the AutoML pipeline, define all hyperparameter space for the pipeline (Data Encoding, Data Imputation, Data Balancing, Data Scaling, Data Feature Selection and Classification/Regression) and an objective function. The objective function just need to carry out one test round of the pipeline and return a evaluation value for the pipeline performance.

The Hyperopt [23] then can carry out a Bayesian Optimization to minimize the objective function (so it's negative model accuracy, or positive mean square error in practice).

The pipeline by default will run 64 evaluation functions (each time run objective function and return a evaluation value) with 6 minutes (360 seconds) limit. If the dataset is large and running 64 evaluations will exceed the time limit (by default or set as another value), the algorithm will complete the current evaluation, stop and find optimal hyperparameter settings based on current best evaluation (in those circumstances, there will not have 64 evaluations). During the optimization process, all hyperparameter settings, preprocessed datasets and evaluation results will be stored in temporary folder for checking.

4 AutoML Pipeline Tests

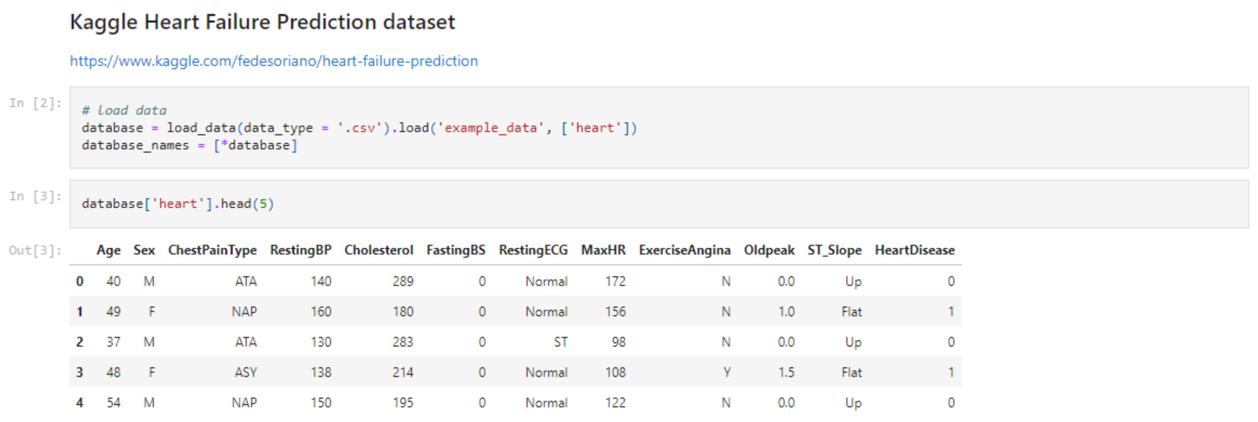
With the AutoML pipeline prepared, it's essential to test the performance of the pipeline on real-life datasets. Here, a classification task on Heart Failure dataset and a regression task on Insurance Premium dataset are presented.

(The following notebooks can be accessed at: https://github.com/PanyiDong/AutoML/tree/master/example)

4.1 Heart Failure Prediction

The dataset can be accessed at: https://www.kaggle.com/fedesoriano/heart-failure-prediction

The dataset intends to predict whether a person will suffer from heart disease based on other 11 features, thus a classification task. The *AutoML* class in the package will automatically assign the task to *AutoClassifier* by the response variable (*HeartDisease* here) AutoML workflow can be simple as read in data (shown in Fig9) and train/test the dataset (shown in Fig10).



Kaggle Heart Failure Prediction dataset

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

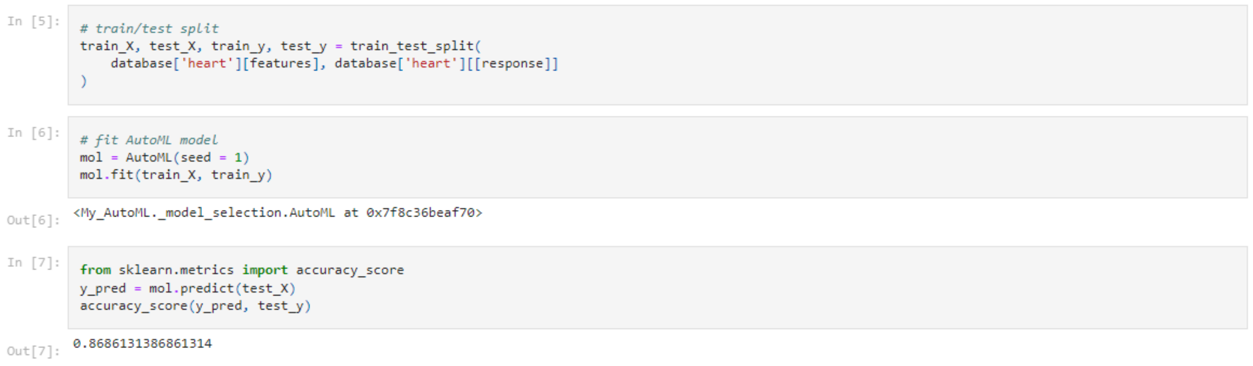
```
In [2]:
# load data
database = load_data(data_type = '.csv').load('example_data', ['heart'])
database_names = [*database]
```

```
In [3]:
database['heart'].head(5)
```

Out[3]:

	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

Figure 9: Read in Heart Failure Dataset



```
In [5]:
# train/test split
train_X, test_X, train_y, test_y = train_test_split(
    database['heart'][features], database['heart'][[response]]
)
```

```
In [6]:
# fit AutoML model
mol = AutoML(seed = 1)
mol.fit(train_X, train_y)

Out[6]: <My_AutoML._model_selection.AutoML at 0x7f8c36beaf70>
```

```
In [7]:
from sklearn.metrics import accuracy_score
y_pred = mol.predict(test_X)
accuracy_score(y_pred, test_y)

Out[7]: 0.8686131386861314
```

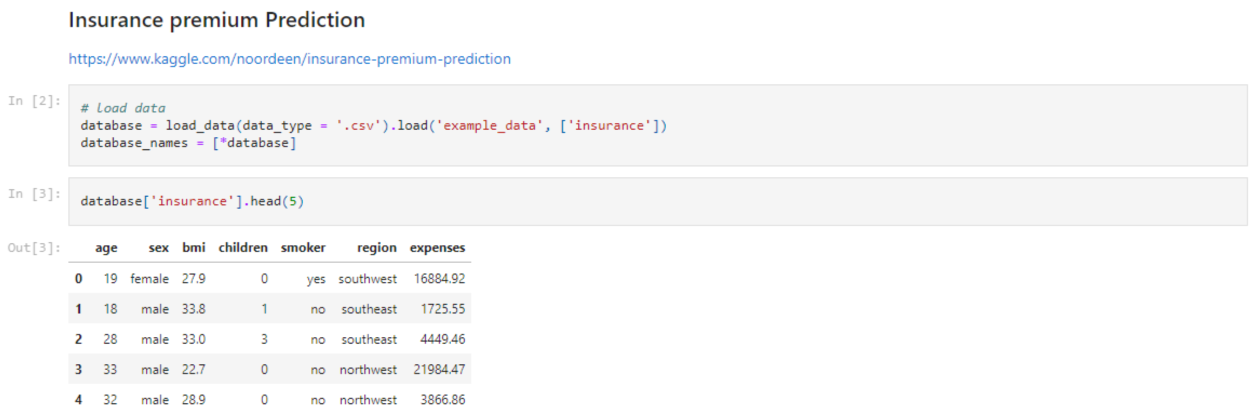
Figure 10: Train/Test on Heart Failure Dataset

As shown in Fig10, the test accuracy is around *0.87*, and the training process takes around 3 minutes. The pipeline is working great for the task both on performance and efficiency.

4.2 Insurance Premium Prediction

The dataset can be accessed at: https://www.kaggle.com/noordeen/insurance-premium-prediction

The dataset intends to predict the insurance premium for some policyholders using 6 other features. Here the task is regression, and *AutoML* class will automatically assign the task to *AutoRegressor* by the response variable (*expenses* here). AutoML workflow can be simple as read in data (shown in Fig11) and train/test the dataset (shown in Fig12).



Insurance premium Prediction

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

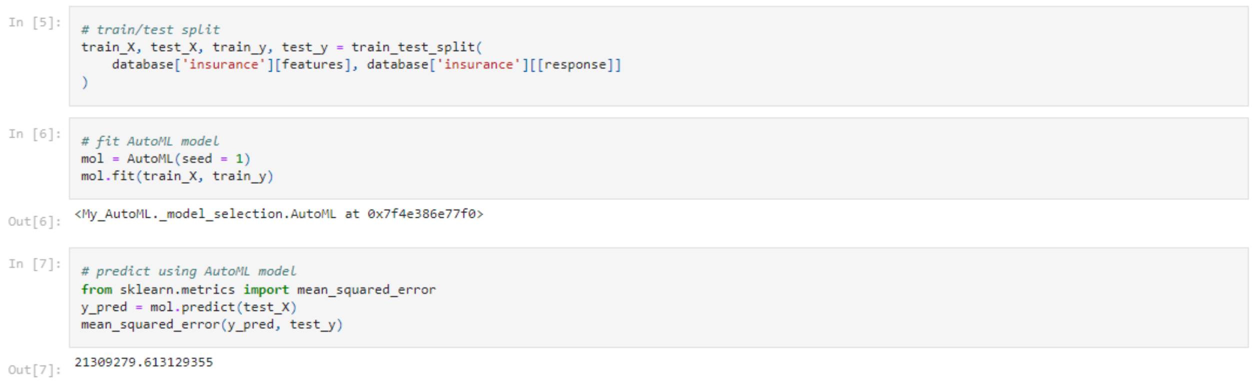
```
In [2]:
# load data
database = load_data(data_type = '.csv').load('example_data', ['insurance'])
database_names = [*database]
```

```
In [3]:
database['insurance'].head(5)
```

Out[3]:

	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

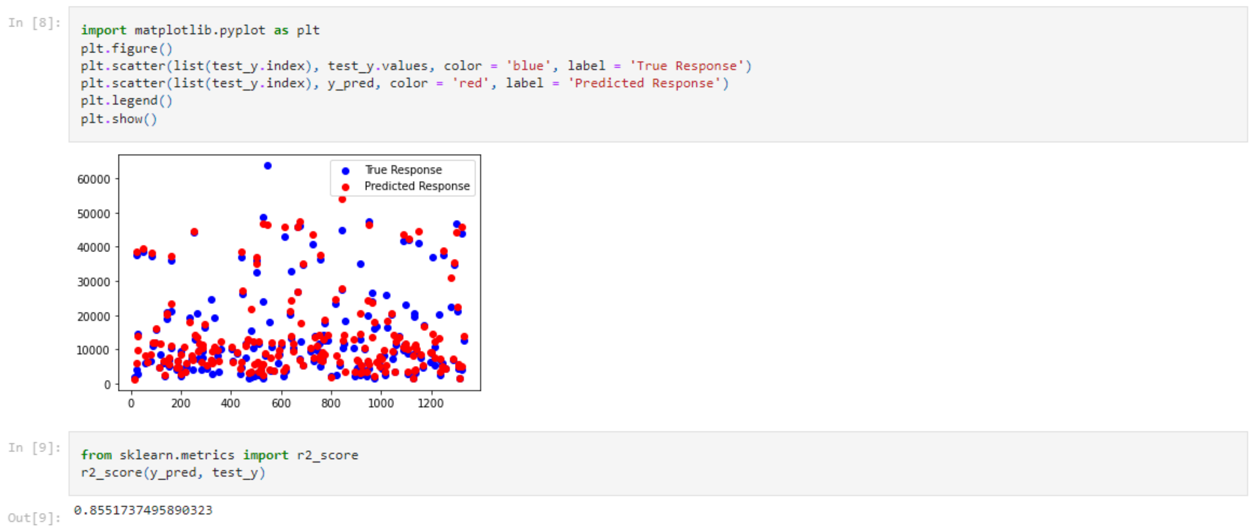
Figure 11: Read in Insurance Premium Dataset



```
In [5]:  # train/test split
         train_X, test_X, train_y, test_y = train_test_split(
             database['insurance'][features], database['insurance'][[response]]
         )

In [6]:  # fit AutoML model
         mol = AutoML(seed = 1)
         mol.fit(train_X, train_y)

Out[6]:  <My_AutoML._model_selection.AutoML at 0x7f4e386e77f0>

In [7]:  # predict using AutoML model
         from sklearn.metrics import mean_squared_error
         y_pred = mol.predict(test_X)
         mean_squared_error(y_pred, test_y)

Out[7]:  21309279.613129355
```

Figure 12: Train/Test on Insurance Premium Dataset

However, a simple MSE evaluation is not enough to recognize whether the AutoML pipeline is performing well or not. So, a further evaluation plotting the true premiums and predicted premiums and R^2 score are shown in Fig13.



```
In [8]:  import matplotlib.pyplot as plt
         plt.figure()
         plt.scatter(list(test_y.index), test_y.values, color = 'blue', label = 'True Response')
         plt.scatter(list(test_y.index), y_pred, color = 'red', label = 'Predicted Response')
         plt.legend()
         plt.show()
```

```
In [9]:  from sklearn.metrics import r2_score
         r2_score(y_pred, test_y)

Out[9]:  0.8551737495890323
```

Figure 13: Further evaluation on Insurance Premium Dataset

As shown in Fig13, the red dots which represent the predicted premiums and blue dots which represent the true premiums demonstrate similar patterns and a 0.85 R^2 score shows that the AutoML pipeline works well.

4.3 Test on Some Datasets

Table1 shows test performance of the AutoML pipeline on some datasets.

Table 1: Test Performance on some datasets

Datasets*	Task Type	Runtime/s	Criterion	Performance**
Employee Future	Classification	142.9	accuracy score	0.8508
Heart Failure	Classification	40.6	accuracy score	0.8686
Insurance Premium	Regression	229.7	mean squared error	21309279.6131
			r2 score	0.8552
Imbalanced Insurance	Classification	over 6000	accuracy score	0.8965
Travel Insurance	Classification	81.9	accuracy score	0.8121
Credit	Classification	256.3	accuracy score	0.982
Medical Premium	Regression	393.1	mean squared error	6927603.5311
			r2 score	0.7912

* Datasets link see Appendix
** The performance of AutML pipeline may varies by random seed, maximum evaluation numbers and time limitation.

5 Summary and Future Improvement

5.1 Summary

The report briefly introduces the components of the AutoML pipeline where the preprocessing procedure consists of Data Encoding, Data Imputation, Data Balancing, Data Scaling, Data Feature Selection and a series of regression/classification models then are selected/tuned to complete the desired task. The Hyperopt package is applied in the pipeline to with Bayesian Optimization to achieve the goal of model selection and hyperparameter optimization (combined step). Then the AutoML pipeline is tested on some real-life datasets, which all reach acceptable performance within relatively short time. Increase on the maximum evaluation numbers can further improve the pipeline's performance (now set 64 evaluations as default). However, the AutoML is still not reaching state-of-art performance on the datasets and efficiency of the pipeline is not ideal (some of the datasets take too long to finish, especially with the increase of dataset size). Currently, the pipeline can served as a baseline for Machine Learning tasks and then further tune the hyperparameters to achieve better performance.

5.2 Future Improvement

The efficiency of the pipeline is not ideal especially with relatively large datasets (Imbalanced Insurance dataset takes storage of around 21 MB, with 300 thousands observations and 12 features, but takes around 100 minutes). Nowadays, the dataset sizes (if images are considered though it's not accepted by the pipeline) can take hundreds of Megabytes, which can not be easily dealt by the pipeline. Of course, there's improvement in the efficiency with further optimizing on the algorithms. However, another direction is to incorporate a parallel computing algorithm with the Model Selection/Hyperparameter Optimization (task parallelism) or the preprocessing steps (data parallelism), so the pipeline can take advantages of multi-core systems.

Currently, only few ML models are available in the pipeline, and all of them are traditional models like tree models, SVM. A incorporation with neural networks may be beneficial for the pipeline's performance. Nowadays, neural networks (or the field of Deep Learning) are more popular and are proved to be successful for various tasks. Networks like Convolutional Neural Networks (CNN), Recurrent Neural Networks (RNN), ResNet, Long Short-Term Memory (LSTM), etc. can be implemented into a AutoML pipeline which can be solved as a problem of Neural Architecture Search (NAS) [24].

References

[1] Müller, A. C. & Guido, S. *Introduction to machine learning with Python: a guide for data scientists* (" O'Reilly Media, Inc.", 2016).

[2] Feurer, M. & Hutter, F. Hyperparameter optimization. In *Automated machine learning*, 3–33 (Springer, Cham, 2019).

[3] Stekhoven, D. J. & Bühlmann, P. Missforest—non-parametric missing value imputation for mixed-type data. *Bioinformatics* **28**, 112–118 (2012).

[4] Azur, M. J., Stuart, E. A., Frangakis, C. & Leaf, P. J. Multiple imputation by chained equations: what is it and how does it work? *International journal of methods in psychiatric research* **20**, 40–49 (2011).

[5] Yoon, J., Jordon, J. & Schaar, M. Gain: Missing data imputation using generative adversarial nets. In *International Conference on Machine Learning*, 5689–5698 (PMLR, 2018).

[6] Batista, G. E., Prati, R. C. & Monard, M. C. A study of the behavior of several methods for balancing machine learning training data. *ACM SIGKDD explorations newsletter* **6**, 20–29 (2004).

[7] Tomek, I. *et al.* Two modifications of cnn. (1976).

[8] Wilson, D. L. Asymptotic properties of nearest neighbor rules using edited data. *IEEE Transactions on Systems, Man, and Cybernetics* 408–421 (1972).

[9] Hart, P. The condensed nearest neighbor rule (corresp.). *IEEE transactions on information theory* **14**, 515–516 (1968).

[10] Chawla, N. V., Bowyer, K. W., Hall, L. O. & Kegelmeyer, W. P. Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research* **16**, 321–357 (2002).

[11] Chandrashekar, G. & Sahin, F. A survey on feature selection methods. *Computers & Electrical Engineering* **40**, 16–28 (2014).

[12] Somol, P., Pudil, P., Novovičová, J. & Paclık, P. Adaptive floating search methods in feature selection. *Pattern recognition letters* **20**, 1157–1163 (1999).

[13] Tan, F., Fu, X., Zhang, Y. & Bourgeois, A. G. A genetic algorithm-based method for feature subset selection. *Soft Computing* **12**, 111–120 (2008).

[14] R0oland. Onepointcrossover. *https://en.wikipedia.org/wiki/File:OnePointCrossover.svg* (2013).

[15] Feurer, M., Klein, A., Eggensperger, J., Katharina Springenberg, Blum, M. & Hutter, F. Efficient and robust automated machine learning. In *Advances in Neural Information Processing Systems 28 (2015)*, 2962–2970 (2015).

[16] Hastie, T., Rosset, S., Zhu, J. & Zou, H. Multi-class adaboost. *Statistics and its Interface* **2**, 349–360 (2009).

[17] Safavian, S. R. & Landgrebe, D. A survey of decision tree classifier methodology. *IEEE transactions on systems, man, and cybernetics* **21**, 660–674 (1991).

[18] 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).

[19] 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).

[20] 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).

[21] Wilhelm, F. The idea behind automatic relevance determination and bayesian interpolation. *https://www.youtube.com/watch?v=2gT-Q0NZzoE* (2016).

[22] Wang, J. An intuitive tutorial to gaussian processes regression. *arXiv preprint arXiv:2009.10862* (2020).

[23] Bergstra, J., Yamins, D. & Cox, D. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In *International conference on machine learning*, 115–123 (PMLR, 2013).

[24] He, X., Zhao, K. & Chu, X. Automl: A survey of the state-of-the-art. *Knowledge-Based Systems* **212**, 106622 (2021).

Appendices

1. Link for Test Datasets

1. Employee Future: https://github.com/PanyiDong/AutoML/blob/master/Appendix/ Employee.csv
2. Heart Failure: https://github.com/PanyiDong/AutoML/blob/master/Appendix/heart.csv
3. Insurance Premium: https://github.com/PanyiDong/AutoML/blob/master/Appendix/ insurance.csv
4. Imbalanced Insurance: https://github.com/PanyiDong/AutoML/tree/master/Appendix/ Imbalanced_Insurance_Data
5. Travel Insurance:https://github.com/PanyiDong/AutoML/blob/master/Appendix/ TravelInsurancePrediction.csv
6. Credit: https://github.com/PanyiDong/AutoML/blob/master/Appendix/credit.rda
7. Medical Premium: https://github.com/PanyiDong/AutoML/blob/master/Appendix/ Medicalpremium.csv