Automated Feature Engineering for AutoML Using Genetic Algorithms

Kevin Shi and Sherif Saad
School of Computer Science, University of Windsor, Sunset Ave, Windsor, Canada

Keywords: Automated Machine Learning, Optimization, Genetic Algorithm, Feature Engineering.

Abstract: Automated machine learning (AutoML) is an approach to automate the creation of machine learning pipelines and models. The ability to automatically create a machine learning pipeline would allow users without machine learning knowledge to create and use machine learning systems. However, many AutoML tools have no or limited automated feature engineering support. We develop an approach that is able to augment existing AutoMl tools with automated feature generation and selection. This generation method uses feature generators guided by and genetic algorithm to generate and select features as part of the AutoMl model selection process. We show that this approach is able to improve the AutoML model performance in 77% of all tested cases with up to 78% error reduction. Our approach explores how existing AutoML tools can be augmented with more automated steps to improve the generated machine learning pipeline's performance.

1 INTRODUCTION

The use of machine learning plays a key role in today's society, with many models providing data to change both the physical and digital world. As with many technologies, automation is a logical next step, but the complex and customizable nature of the machine learning pipeline makes this difficult. In recent years tools such as automated machine learning have automated the process of selecting machine learning models and tuning them. During research on automated machine learning tools, four main weaknesses were identified: feature engineering, model drift, interpretability and explainability, and data quality. This research will study one of the four weaknesses: feature engineering by creating a method of automating feature engineering using genetic algorithms to optimize the transformation of features from their raw state to transformed state for the use of machine learning models. We hope by using a genetic algorithm for automating feature engineering, we can avoid the black-box nature of many current automated machine learning tools as the process of feature engineering will be visible to the user and the generations explainable.

This study investigates the use of genetic algorithms for automated feature engineering to augment AutoML tools. In particular, we are interested in understanding the effects of extending existing AutoML and the impacts on overall tool performance and time requirements. The contribution of this paper is threefold.

1. First, it implements a genetic automated feature engineering tool for AutoML.
2. Second, it compares the performance of different generation approaches for automated feature engineering.
3. Third, it assesses the impact of automated feature engineering for AutoML and traditional machine learning models.

The rest of the paper is structured as follows. Section 2 is a literature review covering existing automated feature engineering methods and the current support for feature engineering in AutoML tools. Section 3 describes the methodology we follow and provides a breakdown of the automated feature engineering method. Section 4 presents our experimental work and discusses our findings. Finally, section 5 concludes our paper and presents potential future work.

2 LITERATURE REVIEW

The existing research in regard to developing automated feature engineering for AutoML can be best divided into two categories, research that focuses purely on automated feature engineering and AutoML tools

Shi, K. and Saad, S.
Automated Feature Engineering for AutoML Using Genetic Algorithms.
DOI: 10.5220/0012090400003555
In *Proceedings of the 20th International Conference on Security and Cryptography (SECRYPT 2023)*, pages 450-459
ISBN: 978-989-758-666-8; ISSN: 2184-7711

with automated feature engineering. The first category of research generally focuses on stand-alone tools that can automate an aspect of feature engineering, which will then be used to augment a machine learning classifier. The second category focuses on AutoML tools with built-in automated feature engineering. These tools have different methods which are used as part of the AutoML optimization and model search. In this section, we aim to give an overview of the existing research in this area.

2.1 Automated Feature Engineering

Feature engineering is a process of selecting and transforming raw features to use in a machine learning model. The automation of feature engineering can be divided based on if the tool is for generalized feature engineering, a tool that is able to work with any input data or specialized for certain problems or data types.

Automated Feature engineering tools that are targeted or specialized range in many forms and can range different types of problems and data types. Some examples of this type of tool include TimeAutoML(Jiao et al., 2020) tailored for time series data with specialized feature generators to tools that target problems such as bus passenger flow(Liu et al., 2021) and customer segmentation(Lee et al., 2021). These tools have a very diverse methodology, from defined feature generators and neural networks to clustering-based approaches. Generally, these types of automated Feature engineering tools are able to generate features that better leverage the data as they are tools tailored to the problem.

The generation-selection approach has been used as part of many automated feature engineering models, these methods differ in the methods of generation and selection, but all contain this common aspect. These models generate transformed features from the data and then apply a method of selection for the generation of the final dataset features.

A tool that follows the generation-selection approach is ExploreKit(Katz et al., 2016) a technique in which features are generated from input data based on defined transformation functions, and after all the features are generated, all features are ranked and then the highest-ranked features are tested. Features that pass the predefined threshold are selected and added to the data to be used to generate new features.

A second tool that follows the generation-selection approach is AutoLearn(Kaul et al., 2017). In this approach, each feature is used to predict the value of other features by applying regression. This process is performed in four steps. Firstly, preprocessing is done by ranking candidate features using information gain, then mining correlated features in which the correlation between features is determined using distance correlation. The next step of the process is feature generation, in which new features are created by regressing every correlated feature pair and by taking the difference between one of the correlated features and the generated feature. Lately, feature selection is applied to all generated features based on feature stability and information gain.

In the research, (Tran et al., 2016) genetic programming is used to create a transformation tree that is able to create defined feature sets. With the transformed features, the dimensional of the data is able to be reduced without reducing classifier performance. Genetic algorithms have also been used to perform automated feature selection as part of a feature engineering model with manually created features selected and optimized with a genetic algorithm(Khan and Byun, 2020).

Evolutionary Algorithms have been used to augment other automated feature engineering approaches such as in (Parfenov et al., 2020). In this approach, the feature generation is performed with the first stage of ExploreKit with genetic optimization being used in place of the original ranking classifier for the selection of original and created features to be added to the output data set.

2.2 Automated Machine Learning (AutoML)

Automated machine learning aims to automate the process of creating machine learning pipelines while requiring minimal human intervention. Current AutoML tools are able to automate some of all of the following steps of the machine learning pipeline: data pre-processing, feature engineering, model selection and hyperparameter optimization. Many AutoML tools focus on model selection and hyperparameter optimization in which a machine learning classifier is picked and optimized for the dataset. Some examples of common AutoML tools include TPOT, autosklearn, FLAML, mljar, AutoKeras, and Autogulon. Out of the six tools, TPOT, mljar, and Autogulon have built-in support for automated feature engineering, while the three other tools lack support and focus on model optimization. Each of the tools that support automated feature engineering has different behaviours.

TPOT works to automate feature engineering as part of the automated pipeline. The genetic programming approach can choose to include feature engineering tools from sklearn, such as Polynomial Fea-

tures. These feature engineering steps become part of the total machine learning pipeline that is optimized by TPOT.

MLJAR performs feature engineering as part of its AutoML optimization process with a module for feature engineering called golden features. With golden features, the tool aims to create new features from the data that have greater predictive power. These features are created by generating all possible feature pairs based on division or subtraction operations and then testing these features by training them on parts of the dataset. MLJAR is able to generate new features as part of its AutoML optimization process.

Autogulon automates feature engineering using defined feature generators that are automatically applied based on data type with an included generator for numeric, categorical, data and text features. These feature generators are applied to features within the dataset before the rest of the machine learning pipeline is optimized.

These three AutoML approaches all support automated feature engineering but very different implementations with approaches applying feature engineering before, during and as part of the optimization process.

Other AutoML approaches, such as Pycaret, have support for featuring engineering steps built into the tool but do not automatically apply these feature engineering steps. For these tools, the user must manually enable the built-in functions.

Overall support for feature engineering in AutoML can be best broken down into three categories automated, manual and without. For automated tools, the scope and impact of feature engineering can be expanded, while for AutoML tools with manual support or without support, an approach that can automate automatic feature engineering such as this research can improve the automation of the creation of the whole machine learning pipeline. External tools that incorporate different types of AutoML tools allow for automated feature engineering to support more AutoML tools versus an add-on developed for a singular tool. This approach has also been seen in literature, such as running an automated feature extraction tool before the AutoML system for a limited degree of success(Eldeeb et al., 2021).

2.3 Data Augmentation

Data augmentation is any technique that changes the input data such that there is an increase in data size or the quality of the data. These approaches are commonly used for image processing but data augmentation can also be applied to any type of data. DeltaPy(Snow, 2020) is a framework for tabular data augmentation, which is a process that allows for modular feature engineering. DeltaPy provides 50 prebuilt augmentation functions from the following categories: transforming, interacting, mapping, extracting, and synthesising. Transforming is the process in which a single input feature produces new features. Interacting is the process in which more than one feature is required to create new features. Mapping functions aim to remove noise from the data or highlight signals improving the overall quality of the data. Overall, DeltaPy provides many functions that can assist with the process of feature engineering for tabular data and allows for a modular approach to feature engineering.

3 METHODOLOGY

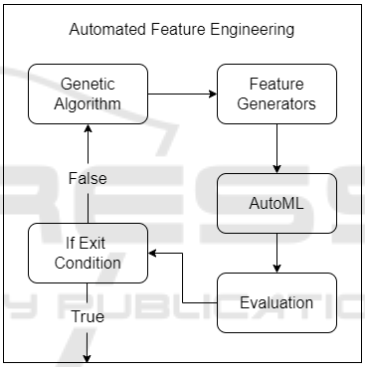

Figure 1: Automated Feature Engineering System Design.

The automated feature engineering system is shown in figure 1. The system follows an iterative approach where each iteration is composed of three main phases: genetic algorithm, feature generators, and AutoML. The genetic algorithm's role is to provide the settings for the feature generators; this includes the type of feature generator and the target features and modifiers to be used. With each iteration, the genetic algorithm works to optimize these settings for the best performance. The feature generators using the setting from the genetic algorithm apply the chosen transformation to the dataset and creates the new candidate feature. The AutoML step takes the candidate feature and the base data and trains an AutoML that will be evaluated based on its accuracy score. The score is used for ranking the generated feature within the genetic algorithm. A portion of the data is held out at the start of the process, which will be used to evaluate overall system performance. Next, we describe each of the main phases in detail

below. This approach, as presented, differs from the currently available AutoML feature engineering approaches. Firstly this approach can augment any AutoML tool as it is not limited to a single tool. Secondly, unlike the approaches presented by AutoGluon and mljar, this approach is iterative and works to optimize for the best possible generated features, versus only running feature generation at the start of the process and a single time as part of the AutoML process, respectively. This approach is similar to TPOT as both are evolutionary algorithm-based approaches, but more flexibility in feature generators is possible in the presented approach.

3.1 Feature Generation

Normal feature generation is performed using feature generation functions. These generation functions can be best split into binary and unary operators. The binary operators are either arithmetic (addition, subtraction, multiplication and division) or boolean operators (less than, is equal and greater than). The unary operators have three categories, scaler, clustering and binning. All binary operators are targeted to two features from the genetic algorithm, while unary operators are targeted at one feature. The modifier value is used to influence the generation function, such as for binning and clustering, as well as picking the arithmetic operator. Each generation function uses the features as input and outputs a new feature to be appended to the input data.

3.1.1 DeltaPy

For feature generation using DeltaPy all transforming, interaction and mapping functions were utilized except the following functions due to incompatibility and issues in the functions: transform triple exponential smoothing, transform naive dec, interact autoregression, interact decision tree disc, interact tech, interact genetic feat, mapper cross lag, mapper a chi, mapper encoder dataset. In total 19 functions were utilized with 13 transforming, 2 interaction and 4 mapping. The genetic algorithm was used to select which functions were chosen as well as the target feature or features and modifier if needed.

3.2 Genetic Algorithm

A genetic algorithm is utilized to search the possible features as there can be multiple best features based on the models used. We also aim to avoid using a brute force approach due to the total possible number of generated features. In the sections below, the configuration of the individual sections of the genetic algorithm is described.

3.2.1 Population

Each chromosome in the genetic algorithm represents a generated feature in the normal case or the selected features for the genetic feature selection case. For the normal case the chromosome (see table 1) is four in length with the first two values representing the chosen features of the raw data, the third value representing the selected transformation function and the last value representing the modifier for the transformation function. For the genetic feature selection case, the chromosome is a binary array with a size equal to the total number of raw and transformed features, with a ”1” representing a chosen feature and a ”0” an unselected feature.

3.2.2 Evaluation and Selection

To evaluate the performance of members of the population, each individual member is chosen and an AutoML model is fitted to that member, this resulting accuracy when optimizing for the accuracy or the AUC when optimizing for error reduction is assigned as the score of the chosen member. If any member of the population is unable to be successfully trained and tested due to an invalid chromosome that member is assigned a score of 0. After all members of the population and been trained and evaluated, all members of the population are sorted based on their scores. The top 50% of the population is selected to create the population for the next generation and the bottom 50% is discarded.

3.2.3 Crossover and Mutation

Crossover and mutation are applied to selected members of the population in order to generate the population for the next generation. Crossover requires a pair of chromosomes called parents, which are used to generate two new chromosomes called children; this is done by using data from parents and switching which parent’s data is used after each crossover point. This process generates two new chromosomes that are mirrors of each other based on the data of the parents. After the new members are generated by the crossover operation, the mutation is then applied. Mutation allows for changes in that data, not from the parents, to maintain genetic diversity. For each value in the chromosomes, if the mutation is triggered, that value is replaced with a random value from the allowable range of values.

3.2.4 Stopping

The genetic algorithm will stop when one of two conditions is met: the set number of generations has passed, or early stopping is triggered. On initiation of the genetic algorithm, the maximum number of generations is set which is the maximum number of generations the genetic algorithm can search. Early stopping this triggered if a set number of generations pass without any improvement to the maximum score.

Table 1: Feature chromosome. N: number of raw features, M: number of transformation functions, C: Max allowed modifier.

Feature A	Feature B	Transformation Function	Modifier
0-N	0-N	0-M	0-C

3.3 Generation Approaches

We evaluate five approaches of the methodology that differ in the number of generated features and the method of their generation.

3.3.1 Singular Generation

Singular generation follows the approach presented previously and generates a single feature optimized by the genetic algorithm.

3.3.2 Group Generation

In group generation, the genetic algorithm's chromosomes are changed to represent a group of features instead of a single feature. This is done by extending the length of the chromosome and having each feature sequentially represented by the chromosome. No other changes are made to the overall methodology.

3.3.3 Pool Generation

In pool generation, before the evaluation stage, all members of the population are split randomly into groups that do not persist between generations. The groups are evaluated and selected based on the overall group performance. The best performing group is recorded and acts in place of the optimized feature. No other changes are made to the overall methodology.

3.3.4 Incremental Generation

In incremental generation, the approach from singular generation is followed, but at the end of each generation, if a feature is found to increase the overall performance of the dataset, that feature is appended to the dataset, and this appended dataset is used for the next generation. The process continues until there is no improvement, then the population is randomized, and the system continues until the set number of generations or early stopping is reached. This approach slowly builds the final feature set adding at most a single feature per generation that improves the overall feature set.

3.3.5 Selection

In selection, an initial set of features of a predefined size is generated. The generated set of features, as well as the original set of features, is selected by the genetic algorithm. In this approach, the chromosome is a bit string the size of combined generated and original features. If the bit value corresponding to a feature is "1" that feature is added to the evaluated feature set, while if the bit value is "0" that feature is discarded. The genetic algorithm in this approach aims to optimize which features should be selected from the combined set of features to create a final feature set. No other changes are made to the overall methodology.

3.4 AutoML

The AutoML tool that was chosen to be augmented with automated feature engineering was Fast Library for Automated Machine Learning (FLAML). This AutoML tool focuses on fast automatic turning of machine learning models which is done using a cost effective hyperparameter optimization algorithm. The algorithm used by FLAML is based on the randomized direct search method FLOW2 which starts from a low cost region exploring the search space while focusing on moving quickly toward the low-loss region while avoiding high cost regions unless necessary. This approach allows FLAML to quickly find suitable configurations for machine learning models.

3.5 Datasets

For evaluations of the automatic feature engineering tool, five datasets from the domain of cybersecurity were selected. These datasets as seen in table 2 represent different domains of cybersecurity from malware detection to spam.

3.5.1 CIC-AndMAL2017

This dataset contains malware samples from four categories: Adware, Ransomware, Scareware and SMS Malware. Following the method described by Noorbehbahani et al.(Noorbehbahani et al., 2019), only the

Table 2: Cybersecurity Datasets.

Reference	Dataset Name	Cybersecurity Problem Domain	Number of Features	Number of Samples
(Lashkari et al., 2018)	CICAndMal2017	Malware Detection	84	10854
(Anderson and Roth, 2018)	EMBER2018	Malware Detection	2381	50000
(Moustafa and Slay, 2015)	UNSW-NB15	Intrusion/Anomaly Detection	47	2540044
(?)	Phishing Websites	Phishing Detection	30	11055
(?)	Spambase	Email Spam Detection	58	4601

ransomware datasets of 10 ransomware families and benign samples were used from the CICAndMal2017 as an evaluation dataset.

3.5.2 Ember 2018

This dataset contains malware features from one million portable execution files after performing static analysis. For evaluation as described in (Galen and Steele, 2020), 25000 benign and malicious samples were selected from both the January and February periods for the training dataset. The testing dataset contains 25000 begin and malicious samples from March to December.

3.5.3 UNWS-NB15

This dataset consists of nine types of attacks: Fuzzers, Analysis, Backdoors, DoS, Exploits, Generic, Reconnaissance, Shellcode and Worms. For evaluation, all four data files were combined to create one dataset no other changes to the data were performed.

3.5.4 Spambase

This dataset consists of spam and non-spam emails. No additional processing was required for this dataset.

3.5.5 Phishing Websites

This dataset consists of legitimate and phishing websites. No additional processing was required for this dataset.

4 EXPERIMENT

4.1 Experiment Setup

In this section, we describe both the hardware and software configuration used to test all of the automated feature engineering approaches with all datasets, experiment data generation is shown in figure 2. Experiments were run on Ubuntu 20.04 LTS on a workstation with 12 cores of 3.7Ghz with 96GB of ram. Python version 3.8 with FLAML version 1.0.7 was used. All data without predefined training and testing datasets were tested with a 75-25 training

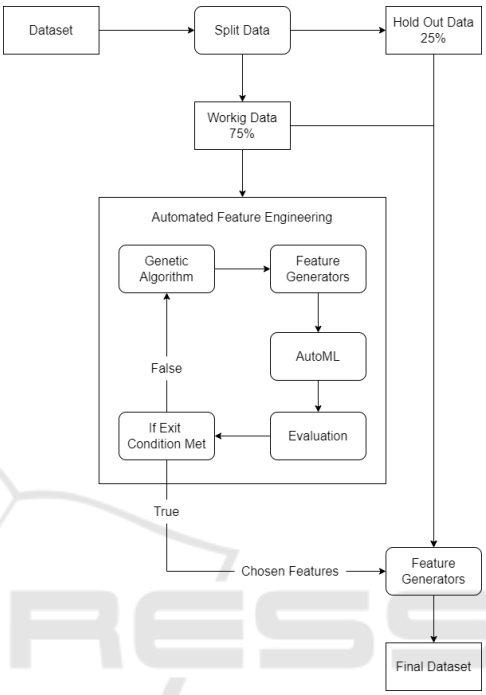

Figure 2: Experiment Data Generation.

to testing split. The maximum training times were set to 600 seconds for Spambase and Phishing Websites, 1800 seconds for CIC-AndMAL2017 and 3600 seconds for Ember and UNWS-NB15. The generic algorithm was set to a maximum of 20 generations with early stopping set to 3, crossover and mutation rates were set to 0.3. The population size for Singular, Group and Incremental was 32, and a group size of 8 was used for Group and Pool generation. The population size for Pool was 256, and the number of generated features for selection was set to 100. For AutoML, within the feature generation process, the maximum time was set to 120 seconds and 10 max iterations.

For comparison between the AutoML (FLAML) and traditional machine learning models, Catboost and three models from scikit-learn, Random Forest, Naive Bayes and Decision Tree were used, with all models set as default. Three evaluations are run for each model, a baseline without any feature engineering, singular generation, and incremental generation.

4.1.1 Metrics

For the evaluation, three metrics are used to measure the performance of the automated feature engineering approaches. The first metric used is accuracy which provides a summary of the given AutoML on the generated feature set by calculating the number of correct predictions divided by the total number of predictions as shown in equation 1. Accuracy is a metric that can be used to show the performance of any given model but has limitations when the data is highly unbalanced.

$$accuracy = \frac{(TP+TN)}{TP+FN+FP+TN} \tag{1}$$

The second metric used was balanced accuracy as another metric to measure the performance of ML models created by the models and tested approaches. Balanced accuracy is a better metric to measure the performance of ML models with imbalanced data. The balanced accuracy is the mean of sensitivity and specificity as shown in equation 2

$$balanced\_accuracy = \frac{TP}{2(TP+FN)} + \frac{TN}{2(TN+FP)} \tag{2}$$

The third metric used in the experiment was error reduction. This metric shows the relative reeducation in error from each of the tested models, which is calculated from the original error minus the new error divided by the original error as shown in equation 3.

$$error\,reduction = \frac{(1-AUC_{original}) - (1-AUC_{new})}{(1-AUC_{original})} \tag{3}$$

4.2 Experiment Results

4.2.1 Generation Approach Comparison

Overall from the experiment results, as seen in table 3, the automated feature engineering tool was able to improve the accuracy of the trained model in the majority of cases, with 26 seeing improvement and 9 with regression in performance. Both the dataset tested and the generation approach of the automated feature engineering tool had major impacts on its performance. Out of the five datasets, two (CICMAL. UNSW) showed improvement with any generation approach, while Phishing Websites improved will all generation approaches except selection. For the Spambase and EMBER datasets, only three generation approaches were able to produce an improvement, with Delta working for Spambase and pool for EMBER. Two generation approaches, Singular and Incremental, were able to improve the accuracy of the data in all tested cases. The balanced accuracy results match the accuracy results except Incremental generation was not able to improve in all cases while Singular generation was able

Looking at the baseline AUC and error reduction as seen in table 4, the automated feature engineering was able to reduce the error of the trained model in the majority of cases for four datasets and reduce the error of the other dataset EMBER in three out of the seven total cases. In total, the tool was able to reduce error in 26 cases and could not, in 9 cases, the same number as the accuracy results. As with accuracy, performance on EMBER was weak, but in this case, Spambase was able to improve in six of the total seven approaches. Delta incremental had the greatest average error reduction of 32% but was only able to provide an error reduction in three of the five datasets. Both singular and group generation were able to reduce error in all five tested datasets, with singular generation having a higher average error reduction of 16% compared to 9%. The dataset that resulted in the most error reduction was CICMAL, with an average of 31%. Unlike with the accuracy results, no dataset resulted in error reduction with all approaches tested, with the best being six out of the seven total approaches. Overall applying the automated feature engineering tool was able to reduce the error in the large majority of tested cases.

The automated feature engineering times are in table 5. The fitting time of the feature engineering tool was impacted by the dataset tested. The effects can be best broken down into two parts feature generation time and AutoML training time. There was no notable impact on overall times for the basic feature generation, with only one case using basic feature generation having a time of greater than one hour. Feature generation using DeltaPy did have an effect on training times for the larger datasets, but this effect depended on which feature generators were chosen, as seen in the difference of time for Delta versus Delta incremental generation for EMBER. As well, feature generation using DeltaPy resulted in increased training time for all datasets except Spambase. Unlike feature generation, the AutoML training time has a fixed upper bound of 120 seconds, this limits the maximum impact of the training time, but the training time still has a major effect on the feature engineer time, especially for the smaller datasets. The generation approach with the lowest average training time was selection which was the least consistent generation approach, only providing improvements in 2 out of 5 datasets. This was then followed by Singular and In-

cremental, the approach with the lowest average performance improvements. In this research, the automated feature engineering process aimed to produce new features quickly. It is possible to increase the genetic algorithm's population, the number of iterations, and the maximum training time for the AutoML tool, which would increase feature engineering times, but it would be expected that higher-performing features would be found. The settings from this study should not be seen as optimized settings for this tool but taken as a possible set of settings to show this proof of concept. It is not the aim of this research to optimize the approach demonstrated.

4.2.2 AutoML and Traditional Machine Learning Comparison

The results can be found in table 6 for accuracy and balanced accuracy. Starting with the accuracy metric, before applying feature engineering, AutoML had the highest accuracy in 2 datasets, UNSW and EMBER, and tied Catboost in Spambase. Catboost leads Phishing Websites and Random Forest had the highest accuracy in CICMAL. While for balanced accuracy lead in 3 datasets Spambase, UNSW and EMBER, while Catboost lead Phishing Websites and random forest lead CICMAL. After applying feature engineering, the changes were that Catboost leads Phishing Websites for accuracy and Random Forest leads for both metrics in UNSW. The method with the highest average accuracy and balanced accuracy was Catboost with singular feature engineering, followed by other Catboost approaches and then the random forest approaches. The Naive Bayes approaches had the lowest average accuracy and balanced accuracy, but feature engineering resulted in the greatest average improvement of 0.076 and 0.055 for balanced accuracy with incremental generation. The approach with the second highest average improvement was AutoML with incremental generation. The AutoML approaches were also the most consistent achieving an accuracy improvement in all ten tested cases for accuracy and 9 out of the 10 cases for balanced accuracy. The random forest and Naive Bayes models archived improvement ins 7 out of the 10 cases for both metrics.

Exploring the error reduction results as seen in table 7, the AutoML approach has the highest starting AUC in 3 of the datasets, Spambase. UNSW and Ember with the random forest model leading in 2, Phishing Websites and CICMAL. After applying feature engineering, the AutoML approach with the incremental generation now leads CICMAL. Looking at the average error reduction, the average performance with AutoML greatly exceeds any other tested approach with almost 16% average error reduction for both singular and incremental generation. The second best approach, random forest singular generation, was only able to achieve and 2.29% average error reduction. The AutoML approach also tied random forest for the most consistent error reduction, reducing the model error in 9 out of the 10 tested cases. The Naive Bayes approaches had poor consistency in error reduction with singular generation resulting in 2 out of 5 and incremental resulting in 1 out of 5 models with error reduction. As well, the Naive Bayes approaches no longer had the greatest average improvement, only achieving an error reduction of 1.24% for singular and 1.64% for incremental.

Overall, for all of the configurations tested and the different metrics used using an AutoML tool resulted in the most consistent improvement, bust as seen form these results the feature engineering tool is able to be applied to other machine learning model with smaller and less consistent improvements possible.

The automated feature engineering generation times are in table 8. Depending on the dataset used and the machine learning model, the automated feature engineering process can be completed in a number of seconds to over an hour. The model used significantly affected the generation time, with approaches such as Catboost requiring more time than any other approach 4812 seconds for singular and 6820 seconds for incremental generation on average. Naive Bayes required the least amount of time on average only 201 seconds for singular and 608 seconds for incremental generation. The only models that required less training time than the original AutoML approach are Naive Bayes and random forest singular generation. This means that the AutoML approach is able to train ten models per candidate in less time than other approaches needed to train one. It would be possible to optimize Catboost for better training time as done with the AutoML approach, but this may impact performance.

5 CONCLUSION AND FUTURE WORK

In this research, we have shown that by utilizing genetic algorithms and feature generators along with AutoML tools, new features can be generated that can boost the performance of classifiers in most tested cases. We have explored different generation approaches as well as different feature generators. From the experiment, we found it is possible to limit the training time of the feature engineering process so that it has an acceptable time requirement. While the impact of certain feature generators can have a much

Table 3: Generation Approach Accuracy and Balanced Accuracy Comparison.

Metric	Accuracy						Balanced Accuracy					
Dataset	Spambase	PhishingWebsites	CICMAL	UNSW	EMBER	Average	Spambase	PhishingWebsites	CICMAL	UNSW	EMBER	Average
Baseline	96.09	96.82	49.92	99.6	93.39	87.164	95.8	96.62	50.4	99.05	93.61	87.096
Singular	96.35	96.89	55.57	99.69	93.5	88.4	95.95	96.71	55.99	99.26	93.7	88.322
Group	95.92	96.89	59.62	99.68	93.15	89.052	95.92	96.64	60.01	99.26	93.38	89.042
Pool	95.74	96.89	59.01	99.7	93.4	88.948	95.34	96.67	59.4	99.3	93.62	88.866
Incremental	96.52	96.85	57.85	99.69	93.4	88.862	96.13	96.62	57.77	99.26	93.62	88.68
Selection	95.57	96.09	59.46	99.65	92.96	88.746	95.19	95.85	59.85	99.19	93.19	88.654
Delta	97.91	97.07	59.01	99.7	93.33	89.404	97.69	96.83	59.4	99.29	93.55	89.352
Delta Incremental	96	98.84	59.73	99.7	93.32	89.518	95.72	98.72	60.11	99.29	93.53	89.474
Average	96.2625	97.0425	57.5213	99.6763	93.3063		95.991429	96.86285714	58.9329	99.2643	93.5129	

Table 4: Generation Approach Comparison AUC Error Reduction.

	Spambase	PhishingWebsites	CICMAL	UNSW	EMBER	Average
Baseline	0.9909	0.9964	0.8475	0.9999	0.9867	0.9643
Singular	4.84%	15.60%	41.45%	17.73%	0.11%	15.95%
Group	5.97%	4.35%	25.90%	5.03%	1.62%	8.57%
Pool	2.93%	13.65%	-4.70%	7.63%	-0.27%	3.85%
Incremental	3.27%	14.17%	66.56%	-4.69%	0.33%	15.93%
Selection	-24.97%	5.86%	37.09%	16.91%	-12.50%	4.48%
Delta	73.12%	1.12%	9.72%	-6.04%	-7.15%	14.15%
Delta Incremental	78.07%	-3.27%	74.35%	8.49%	0.00%	31.53%
Average	17.91%	6.44%	31.30%	5.63%	-2.23%	

Table 5: Generation Approach Comparison Feature Engineering Times (seconds).

	Spambase	PhishingWebsites	CICMAL	UNSW	EMBER	Average
Singular	142.0343898	54.7885598	863.4260866	641.6705883	1970.88141	734.560207
Group	261.404541	114.6803781	1712.215061	3868.684788	2443.341906	1680.065335
Pool	204.7417284	49.2225681	2143.900624	2250.823053	1802.228511	1290.183297
Incremental	286.8734678	83.3469802	2009.662164	1347.980646	1841.885889	1113.949829
Selection	87.9470995	29.2978291	685.3757697	1479.616277	688.2061595	594.088627
Delta	106.6486528	108.726423	1734.906475	10244.63185	1869.085556	2812.799792
Delta Incremental	143.2637108	353.8791434	2853.830053	34302.44831	22799.88297	12090.66084
Average	176.1305129	113.4202688	1714.759462	7733.693645	4773.644629	

Table 6: AutoML and Traditional Machine Learning Accuracy and Balanced Accuracy Comparison.

Metric		Accuracy						Balanced Accuracy					
Model	Method	Spambase	PhishingWebsites	CICMAL	UNSW	EMBER	Average	Spambase	PhishingWebsites	CICMAL	UNSW	EMBER	Average
FLAML	Baseline	96.09	96.82	49.92	99.6	93.39	87.164	95.8	96.62	50.4	99.05	93.61	87.096
FLAML	Singular	96.35	96.89	55.57	99.69	93.5	88.4	95.95	96.71	55.99	99.26	93.7	88.322
FLAML	Incremental	96.52	96.85	57.85	99.69	93.4	88.862	96.13	96.62	57.77	99.26	93.62	88.68
Catboost	Baseline	96.09	96.85	74.61	99.46	91.88	91.778	95.73	96.64	74.62	98.71	92.08	91.556
Catboost	Singular	96.26	97.07	74.79	99.41	91.94	91.894	95.91	96.85	74.79	98.57	92.14	91.652
Catboost	Incremental	96.18	96.78	74.56	99.47	91.82	91.762	95.78	96.59	74.57	98.72	92.03	91.538
Random Forest	Baseline	95.57	96.67	76.65	99.59	88.1	91.316	95.07	96.42	76.65	99.04	88.58	91.152
Random Forest	Singular	95.74	96.71	76.75	99.65	88.02	91.374	95.25	96.49	76.75	99.2	88.51	91.24
Random Forest	Incremental	95.48	96.71	76.83	99.7	87.75	91.294	94.96	96.49	76.84	99.34	88.26	91.178
Naive Bayes	Baseline	82.71	58.25	50.11	83.27	50.95	65.058	84.5	63.48	50.55	84.48	54.18	67.438
Naive Bayes	Singular	83.32	66.32	50.17	82.84	50.95	66.72	85.08	70.54	50.61	84.55	54.18	68.992
Naive Bayes	Incremental	82.97	90.74	50.16	88.49	50.95	72.662	84.75	90.86	50.6	84.38	54.18	72.954
Decision Tree	Baseline	92.18	95.48	74.04	99.49	82.73	88.784	92	95.3	74.04	98.86	83.09	88.658
Decision Tree	Singular	90.18	95.66	73.98	99.49	82.38	88.338	90.01	95.52	73.98	98.86	82.77	88.228
Decision Tree	Incremental	91.31	95.48	74.07	99.49	82.48	88.566	91.01	95.36	74.07	98.86	82.87	88.434

Table 7: AutoML and Traditional Machine Learning Comparison AUC Error Reduction.

Model	Method	Spambase	PhishingWebsites	CICMAL	UNSW	EMBER	Average
FLAML	Baseline	0.9908689	0.996400354	0.847515322	0.999915559	0.9867202	0.964284058
FLAML	Singular	4.84%	15.60%	41.45%	17.73%	0.11%	15.95%
FLAML	Incremental	3.27%	14.17%	66.56%	-4.69%	0.33%	15.93%
Catboost	Baseline	0.9885191	0.997650741	0.846164499	0.999827772	0.9788337	0.962199151
Catboost	Singular	2.61%	-2.39%	0.06%	-5.96%	2.73%	-0.59%
Catboost	Incremental	5.45%	0.09%	-0.52%	-1.17%	2.77%	1.32%
Random Forest	Baseline	0.9859878	0.997714392	0.85519875	0.999854094	0.9654172	0.960834441
Random Forest	Singular	1.91%	3.21%	3.07%	0.65%	2.62%	2.29%
Random Forest	Incremental	7.22%	2.33%	0.74%	-10.05%	0.20%	0.09%
Naive Bayes	Baseline	0.9468869	0.970074923	0.52782945	0.887433898	0.6608854	0.798622115
Naive Bayes	Singular	2.26%	3.94%	0.00%	0.00%	0.00%	1.24%
Naive Bayes	Incremental	-0.01%	8.14%	0.00%	0.00%	0.00%	1.62%
Decision Tree	Baseline	0.915775	0.977057193	0.739819588	0.988409814	0.8296209	0.890136486
Decision Tree	Singular	1.82%	-3.00%	0.31%	0.55%	0.51%	0.04%
Decision Tree	Incremental	-2.44%	-6.58%	0.43%	3.48%	0.65%	-0.89%

Table 8: AutoML and Traditional Machine Learning Comparison Feature Engineering Times (seconds).

Type	Approach	Spambase	PhishingWebsites	CICMAL	UNSW	EMBER	Average
FLAML	Singular	142.0343898	54.7885598	863.4260866	641.6705883	1970.88141	734.560207
FLAML	Incremental	286.8734678	83.3469802	2009.662164	1347.980646	1841.885889	1113.949829
Catboost	Singular	286.7277985	1084.675602	4903.853139	9465.920835	8318.547868	4811.945049
Catboost	Incremental	323.6878317	2189.710747	5479.15481	18088.45976	8020.990814	6820.400792
RF	Singular	25.6818658	37.241105	514.3558151	2183.534547	278.2760238	607.8178714
RF	Incremental	35.674479	39.5919785	771.8287661	9963.641372	513.4612354	2264.839566
NB	Singular	1.1899117	4.7808542	81.4093617	754.2586317	160.8956941	200.5068907
NB	Incremental	2.7123099	4.4873749	140.2050486	2982.596857	186.0150966	663.2033374
DT	Singular	5.6072376	3.8393552	494.7855521	1823.740664	6042.433405	1674.081243
DT	Incremental	6.7526271	3.9819257	1820.335759	1653.910428	5636.691553	1824.334459

greater impact on overall feature engineering time. We show that this approach is also able to support traditional machine learning models, but the AutoML tool benefits more from automated feature engineering. We also show that with the correct optimization, the feature engineering time for AutoML can be less on average than the traditional machine learning models.

The research, as presented, aims to present a proof of concept of using genetic algorithms for feature engineering with AutoML tools. Possible future work areas include custom feature generators tailored to each dataset or problem type. In addition, the inclusion of training time into the genetic search scope would involve allowing the genetic algorithms to set the training time of the AutoML tool while also including the resting training as part of the fitness function. Finally, it would be possible to expand this research by utilizing multiple tools in the training process.

REFERENCES

Anderson, H. S. and Roth, P. (2018). EMBER: An Open Dataset for Training Static PE Malware Machine Learning Models. *ArXiv e-prints*.

Eldeeb, H., Amashukeli, S., and El Shawi, R. (2021). *An Empirical Analysis of Integrating Feature Extraction to Automated Machine Learning Pipeline*, pages 336–344.

Galen, C. and Steele, R. (2020). Evaluating performance maintenance and deterioration over time of machine learning-based malware detection models on the ember pe dataset. In *2020 Seventh International Conference on Social Networks Analysis, Management and Security (SNAMS)*, pages 1–7. IEEE.

Jiao, Y., Yang, K., Dou, S., Luo, P., Liu, S., and Song, D. (2020). Timeautoml: Autonomous representation learning for multivariate irregularly sampled time series.

Katz, G., Shin, E. C. R., and Song, D. (2016). Explorekit: Automatic feature generation and selection. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*, pages 979–984.

Kaul, A., Maheshwary, S., and Pudi, V. (2017). Autolearn — automated feature generation and selection. In *2017 IEEE International Conference on Data Mining (ICDM)*, pages 217–226.

Khan, P. W. and Byun, Y.-C. (2020). Genetic algorithm based optimized feature engineering and hybrid machine learning for effective energy consumption prediction. *IEEE Access*, 8:196274–196286.

Lashkari, A. H., Kadir, A. F. A., Taheri, L., and Ghorbani, A. A. (2018). Toward developing a systematic approach to generate benchmark android malware datasets and classification. In *2018 International Carnahan Conference on Security Technology (ICCST)*, pages 1–7. IEEE.

Lee, Z.-J., Lee, C.-Y., Chang, L.-Y., and Sano, N. (2021). Clustering and classification based on distributed automatic feature engineering for customer segmentation. *Symmetry*, 13(9).

Liu, Y., Lyu, C., Liu, X., and Liu, Z. (2021). Automatic feature engineering for bus passenger flow prediction based on modular convolutional neural network. *IEEE Transactions on Intelligent Transportation Systems*, 22(4):2349–2358.

Moustafa, R. and Slay, J. (2015). A comprehensive data set for network intrusion detection systems. *School of Engineering and Information Technology University of New South Wales at the Australian Defense Force Academy Canberra, Australia, UNSW-NB15*.

Noorbehbahani, F., Rasouli, F., and Saberi, M. (2019). Analysis of machine learning techniques for ransomware detection. In *2019 16th International ISC (Iranian Society of Cryptology) Conference on Information Security and Cryptology (ISCISC)*, pages 128–133. IEEE.

Parfenov, D., Bolodurina, I., Shukhman, A., Zhigalov, A., and Zabrodina, L. (2020). Development and research of an evolutionary algorithm for the formation of a feature space based on automl for solving the problem of identifying cyber attacks. In *2020 International Conference Engineering and Telecommunication (En&T)*, pages 1–5.

Snow, D. (2020). Deltapy: A framework for tabular data augmentation in python. *Available at SSRN 3582219*.

Tran, B., Xue, B., and Zhang, M. (2016). Genetic programming for feature construction and selection in classification on high-dimensional data. *Memetic Computing*, 8(1):3–15.