Stacking Ensemble for auto_ml

Khai T. Ngo

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science
in
Computer Engineering

Joseph M. Ernst, Co-chair
Pratap Tokekar, Co-chair
Robert P. Broadwater

May 1, 2018
Blacksburg, Virginia

Keywords: Machine learning, Stacking Ensemble, Model Selection, Hyper-parameter optimization, auto_ml

Stacking Ensemble for auto_ml

Khai T. Ngo

(ABSTRACT)

Machine learning has been a subject undergoing intense study across many different industries and academic research areas. Companies and researchers have taken full advantages of various machine learning approaches to solve their problems; however, vast understanding and study of the field is required for developers to fully harvest the potential of different machine learning models and to achieve efficient results. Therefore, this thesis begins by comparing auto_ml with other hyper-parameter optimization techniques. auto_ml is a fully autonomous framework that lessens the knowledge prerequisite to accomplish complicated machine learning tasks. The auto_ml framework automatically selects the best features from a given data set and chooses the best model to fit and predict the data. Through multiple tests, auto_ml outperforms MLP and other similar frameworks in various datasets using small amount of processing time.

The thesis then proposes and implements a stacking ensemble technique in order to build protection against over-fitting for small datasets into the auto_ml framework. Stacking is a technique used to combine a collection of Machine Learning models' predictions to arrive at a final prediction. The stacked auto_ml ensemble results are more stable and consistent than the original framework; across different training sizes of all analyzed small datasets.

Stacking Ensemble for auto_ml

Khai T. Ngo

(GENERAL AUDIENCE ABSTRACT)

Machine learning is a concept of using known past data to predict unknown future data. Many different industries uses machine learning; hospitals use machine learning to find mutations in DNA, online retailers use machine learning to recommend items, and advertisers use machine learning to show interesting ads to viewers. With increasing adoption of machine learning in various fields, there are a significant number of developers who want to take advantages of this concept, but they are not deeply familiar with techniques used in machine learning. This thesis introduces auto_ml framework which reduces the required deep understanding of these techniques. auto_ml automatically selects the best technique to use for each individual process, which used to train and predict given datasets. In addition, the thesis also implements a stacking ensemble technique which helps to yield consistently good predictions on small datasets. As the result, auto_ml performs better than MLP and other frameworks. In addition, auto_ml with the stacking ensemble technique performs more consistently than auto_ml without the stacking ensemble technique.

Dedication

I dedicate this thesis to my parents who have given up their dream for me to pursue mine.

Acknowledgments

I would like express my sincere gratitude to Dr. Joseph Ernst and Mr. Michael Fowler for tremendous supports and guidelines throughout my graduate career. I also want to thank the rest of my committee members: Dr. Pratap Tokekar, and Dr. Robert Broadwater.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

Machine learning uses a variety statistical techniques or models to allow computers to learn patterns in data without human guidance. The goal is for computers to predict future events based on the data that the computers have already seen. Machine learning has been used in many applications across multiple different industries. [13] uses machine learning to search for transcription start sites (TSSs) in a genome sequences by learning the past data of TSSs. [12] trains machine learning models using different known hand-written digits to predict unclassified hand-written digits. Machine learning can be divided into two categories: Supervised and Unsupervised machine learning. [13] and [12] use supervised machine learning where the data used to train the models are labeled. Labeled data consists of an input object and a corresponding output. For example, the training data for [12] consists of vectors of pixels that make up a hand-written digit and the actual numbers presented by the hand-written digits. On the other hand, unsupervised machine learning uses unlabeled data. Data are often collected without labels; therefore using supervised machine learning adds the additional overhead of labeling the data. This thesis uses concepts of supervised machine learning and focuses entirely on labeled datasets.

In recent years, automating the design of machine learning frameworks has become very popular. Hyper-parameter optimization, a major component of the design, has proven to

meet and exceed human performance [3]. Hyper-parameter optimization is technique used to let computers learn the most optimized set of parameters for a particular machine learning model. In addition, these frameworks are flexible to different types of datasets (e.g. regression and classification). Therefore, this allows developers to take full advantage of machine learning models and tools without having deep knowledge of machine learning. These frameworks can introduce the benefits of utilizing machine learning to those who are not in the machine learning community.

With the emergence of big data, small datasets have been overlooked by the vast majority of developers; however, there are crucial small data sets that can be of use to solving real life problems. For example, [7] utilized a small dataset for genetic mutation research to detect cancer. Nonetheless, small data sets are problematic when building machine learning models. A common problem that occurs is over-fitting [11]: when a model fits training data too well, learning the details as well as the noises of the data. As a result, over-fitting negatively impacts the performance of the model on testing data. Most of the time, some techniques are used to generate artificial data [1, 7] in order to avoid over-fitting in small datasets. These techniques are consequently domain specific and must be developed for each type of data set. It is therefore infeasible for an automatic machine learning framework to implement such techniques. This thesis presents a stacking implementation to prevent the over-fitting problem of small datasets for auto_ml, an automatic machine learning framework.

1.2 Multiple Layer Perceptron

To evaluate the performance of auto_ml, this thesis compares the framework to a Multiple Layer Perceptron (MLP) network. MLP is fast and often makes very good predictions for large datasets. MLP consists of an input layer, an output layer, and an arbitrary number

of hidden layers. Each layer also includes an arbitrary number of neurons. Each neuron has an output value and connects to all neurons of adjacent layers. A connection between two neurons carries a weight value. When training on MLP, sample data points are evaluated to determine the predicted value and therefore the prediction error, this is called forward pass. (1.1) is used to compute the value of each neuron, where $\hat{v}$ is the new value of a neuron, b is a bias value, n is the total number of neurons connect to the targeted neuron from the previous layer, v_i is the value of ith neuron, and w_i is the weight from ith neuron to the targeted neuron.

$$\hat{v} = b + \sum_{i=1}^{n} (v_i * w_i) \tag{1.1}$$

The output of the neuron is calculated by applying an activation function to the value. There are different functions used for this purpose; some popular functions are the logistic sigmoid, the hyperbolic tangent, and the rectified linear unit function. This thesis uses the logistic sigmoid function (1.2) as the activation function for MLP. The goal of the sigmoid function is converting the $\hat{v}$ to a value between 0 and 1. These set of equations are used to calculate neurons of all layers except for the input layer.

$$f(x) = \frac{1}{(1 + e^x)} \tag{1.2}$$

MLP then uses a back-propagation algorithm to adjust the network according to the prediction error [24] throughout each iteration. The back-propagation algorithm used in this paper is Stochastic Gradient Descent (SGD) [19]. (1.3) shows how MLP uses SGD to update the weights, where $\hat{w}$ is the new weight, w is the previous weight, ρ is the learning rate which limits how quickly the weight of each neuron converges to a prediction error, α and $R(w)$ are the regularization value and functions which will be discussed later in the thesis, and L is

the function used to calculate a prediction error of MLP's predictions to the actual outputs.

$$\hat{w} = w - \rho(\frac{\partial L}{\partial w} + \alpha\frac{\partial R(w)}{\partial w}) \tag{1.3}$$

The prediction error can be calculated using many different equations. This thesis focuses on the scikit-learn neural network implementation, which uses cross-entropy for classification and square error for regression as the loss function [19]. (1.4) and (1.5) show cross-entropy and square error are calculated, respectively; where $\hat{y}$ is the predicted value of all output neurons, y are the observed values, and n is the total number of neurons in the output layers.

$$L(\hat{y}, y) = -y\ln(\hat{y}) - (1-y)\ln(1-\hat{y}) + \alpha R(w) + \tag{1.4}$$

$$L(\hat{y}, y) = \sum_{i=1}^{n}(\hat{y}_i - y_i)^2 + \alpha R(w) \tag{1.5}$$

MLP iterates through the forward pass and SGD for a set number of times or until a certain prediction error is achieved. All of these choices of functions and parameters are part of the design of the neural network. With a vast number of parameters to use, MLP requires a big amount of input from experts to chose and tune the right parameters. In addition, the parameters also need to be optimized specifically for a particular application to yield the best predicting results. That is why the use of hyper-parameter optimization techniques is becoming very popular.

The thesis discusses the use of two different techniques to avoid over-fitting for MLP: regularization and dropout. Regularization prevents over-fitting by adding a penalty to the loss function. Two popular algorithms for regularization are L1 and L2 regularization [18]. The main difference between these algorithms is that L2 uses the sum of the square of weights, (1.6), going to each neuron, while L1 uses just the sum of the weights, (1.7). In the two

equations, n is the number of neurons in the previous layer and w is the weights from the previous nodes to the current node.

$$R(w) = \frac{1}{2}\sum_{i=1}^{n}(w_i^2) \tag{1.6}$$

$$R(w) = \frac{1}{2}\sum_{i=1}^{n}(|w_i|) \tag{1.7}$$

Both L1 and L2 can be added during the back propagation process and loss functions; they also have a multiplier coefficient, α, to control how their penalty affects the loss function as seen in (1.3), (1.4), and (1.5).

Dropout was recently introduced and it is used widely to prevent over-fitting[23]. The concept of dropout is to discard outputs of the neurons in the hidden layers in each iteration during training, based on a certain probability. Fig. 1.1 shows an example of the training process of a normal MLP and an MLP with dropout. For the normal MLP, training phase of each iteration has the same model structure. However, with dropout, MLP discards random neuron using a certain probability in each hidden layer throughout different iterations. In each iteration, both models perform forward pass and back propagation to update all existing weights. After training, the MLP model with dropout uses all neurons to generate predictions; however, weights going to each hidden layer's neurons are reduced by the probability from that hidden layer.

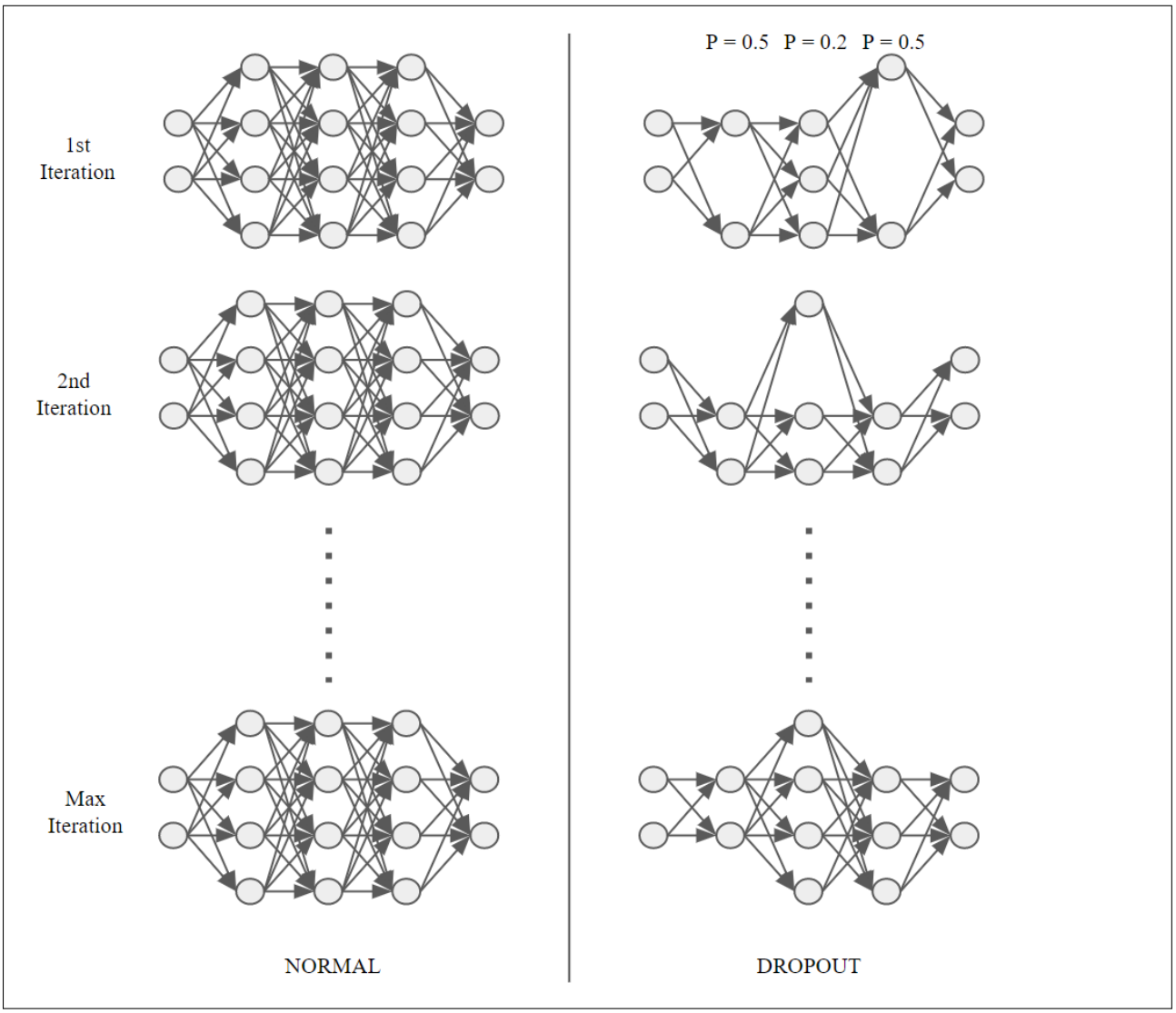

Figure 1.1: Training Phase of a normal MLP model and MLP model with dropout. Initially, both MLP models have identical structures: a input layer, three hidden layers, and a output layer. Input and output layers have 2 neurons each. Each hidden layer has 4 neurons each.

Chapter 2

Related Works

2.1 Hyper-parameter Optimizations

There are different techniques used for hyper-parameter optimization: grid search, random search, and Bayesian optimization. Grid search is a technique used to exhaustively search through a hyper-parameter space for an optimal set. This method was widely used for hyper-parameter optimization until random search was introduced. Random search also performs similar task; however, it includes a random distribution of each hyper-parameter instead searching the entire hyper-parameter space.

Random search picks random set of hyper-parameter and use it to validate the performance of a model. [2] showed that exhaustively searching through a big set of hyper-parameter is much slower than random search and yield similar performance.

Lastly, Bayesian optimization [22] is a technique that utilizes Gaussian Process (GP) to search through a hyper-parameter space, randomly pick, and choose the most optimal set of hyper-parameters. GP is used to calculate a posterior distribution after a set of hyper-parameters is validated. As the validated sets of hyper-parameters grows, the posterior distribution improves, and Bayesian optimization becomes more certain of which regions in the hyper-parameter space are more likely to yield better results. This allows Bayesian optimization to converge faster to the optimal set than random search. For a big hyper-parameter

space, random search and Bayesian optimization are the better choices for hyper-parameter optimization. However, for small hyper-parameter space, grid search is still reliable [2]. In addition, grid search is a better choice for parallelization. Therefore this thesis, which focuses on small dataset, utilizes grid search for hyper-parameter optimization.

2.2 Machine Learning Frameworks

There are several automatic machine learning frameworks that are used today to solve different challenges. Auto-WEKA is an automatic machine learning implementation of the Waikato Environment for Knowledge Analysis (WEKA), an award-winning open-source workbench containing different machine learning models and tools [10]. Auto-WEKA uses Bayesian optimization to search for an optimal machine learning model to train and predict, given a dataset. Auto-sklearn [20] is another framework, which is built from the scikit-learn library. Auto-sklearn also uses Bayesian optimization to search for good machine learning models and then uses weight ensemble [5] to combine predictions from those models.

2.3 Ensembles

Stacking [26] is an ensemble technique that has been becoming very popular among the research community. Stacking is an efficient technique in which the predictions, generated from various different machine learning model, are used as inputs in a meta-learner, a second-layer machine learning model. Unlike other ensemble combination rules [17], which are used just to combine predictions of different models, the meta-learner learns how to combine the predictions. Therefore, it provides a specific and unique way to combine predictions across multiple datasets and produce an efficient results without the need of tuning different en-

semble combination rules. That is why stacking is chosen to be implemented in auto_ml. In fact, with some modifications, stacking has shown to outperform other ensemble techniques. [8] improves the performance by using multi-response model trees at the meta-learner level. [6] uses a popular meta-heuristic approach, Ant Colony Optimization, to outperform conventional ensemble techniques.

Besides stacking, there are two other main ensemble techniques: Bagging and Boosting. Bagging [15], or Bootstrap Aggregating, is a technique that uses combinations with repetitions to produce multiple data subsets of the same cardinality and size to the original data. The way Bagging generates subsets of the original data is call bootstrapping. These subsets are used to train different machine learning models. During predicting phase, predictions of all models are combined using one of many ensemble combination rules [17]. Listed are some of the ensemble combination rules that can be used: majority vote rule, mean rule, product rule, or weighted sum rule. Fig. 2.1 shows how Bagging is used to predict a data point. Bagging increases the training data size to lower the variance of a model's prediction; however, bagging does not improve the model's performance when the training data has low variance [4].

Boosting [15] is very similar to Bagging, but with a slight modification to the process. Instead of generating all subsets at the same time, Boosting generates one subset at a time. The first subset is generated in the same way as Bagging. It is then used to train a model and is tested against the original data. The trained model is used to collect mislabeled data points or data points with high prediction error. These data points combined with other random data points taken from the original data will make up the next subset; Fig. 2.2 shows the Boosting training process. Finally, the models' predictions are combined in the same process as Bagging. As of a result, Boosting improves the overall prediction performance, but it can increase the variance and over-fit the data.

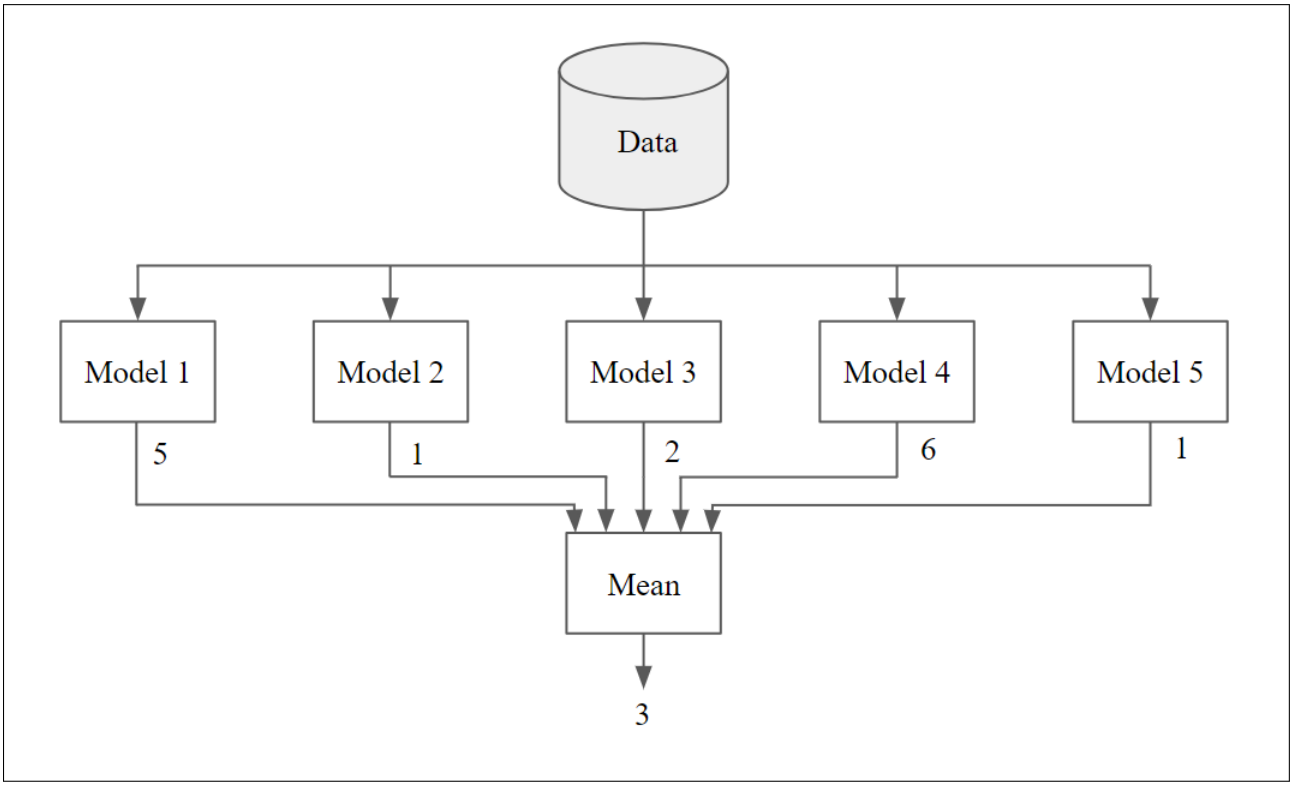

Figure 2.1: Example of how Bagging is used to get a prediction given a data point. In the example, there are five trained train models which are used to generate predictions. These five models are trained using training data subsets. This example uses mean rule to combine the predictions from the five models.

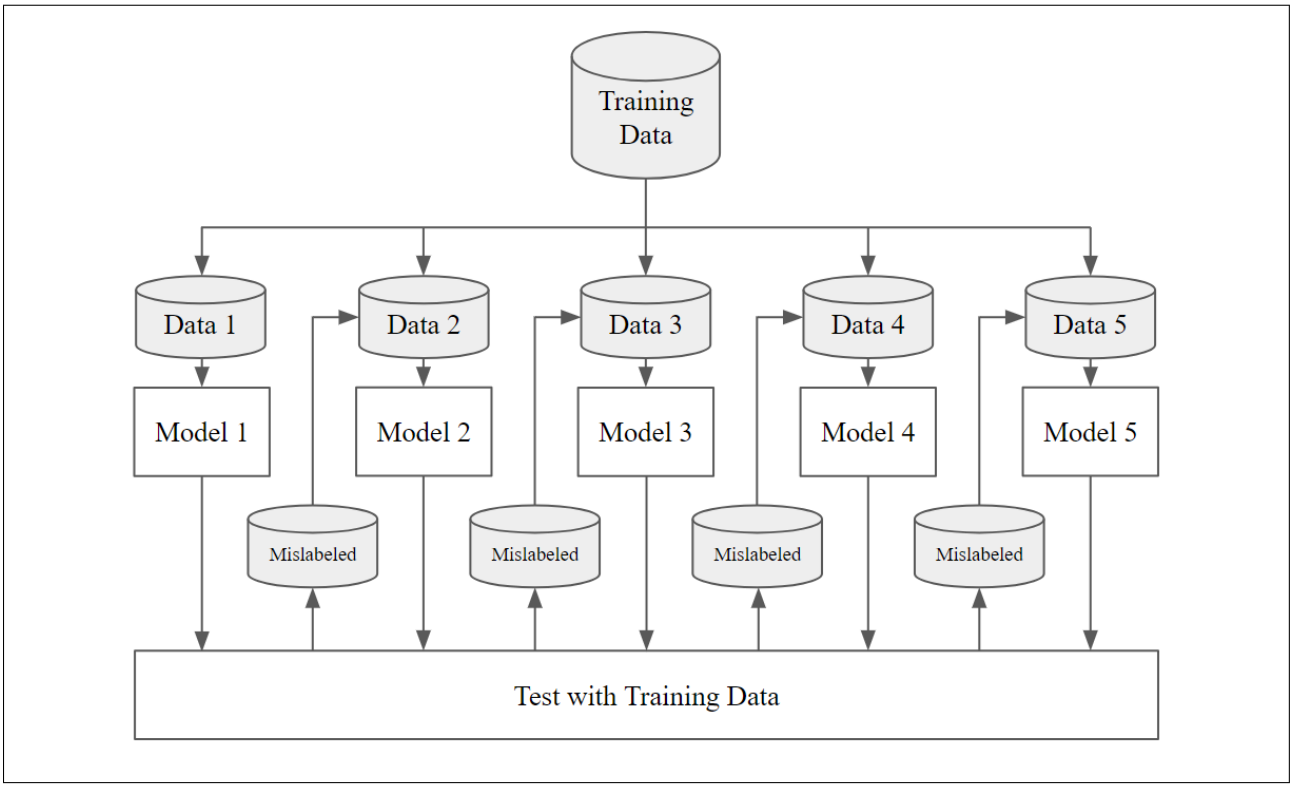

Figure 2.2: Boosting's Training Process. This example has 5 models, each model is trained by a subset data above it. After each model is trained, it is used to generate prediction for the whole training data and some mislabeled data are collected. These mislabeled data together with the next subset data used to train the next model.

Chapter 3

auto_ml

The auto_ml framework is implemented using the scikit-learn library [19]. This library provides a various number of machine learning models, ranging from linear to non-linear models. In addition, it also includes convenient and effective tools for various machine learning processes, such as data pre-processing and hyper-parameter optimization. Compared to other frameworks, auto_ml provides fewer machine learning models and tools; however, on small datasets, it can produce very close performance to other frameworks in just a fraction of the processing time.

3.1 Machine Learning Models

The auto_ml framework has hyper-parameter optimization which chooses different models to fit best for the data. By default, the framework includes four models for regression problems: Gradient Boosting Regression model, Random Forest Regression model, Linear Regression model, and Extra Trees Regression model. And for classification problems, the framework includes: Gradient Boosting Classification model, Logistic Regression model, and Random Forest Classification model. Each model has different advantages and disadvantages; and oftentimes, one model's advantages addresses another models' disadvantages. Table 3.1 lists the advantages and disadvantages of these models on four factors: accuracy of the predictions, training runtime, and what is the likelihood of each model to overfit as well as

underfit. Under-fitting is the opposite of over-fitting, when complex datasets are trained on a simple model. Therefore, the model can not fit the complicated relationship of features and their output; resulting in poor performance.

Model	Accuracy	Runtime	Overfitting	Underfitting
Linear Models	medium	very fast	least likely	most likely
Gradient Boosting	very high	medium	most likely	least likely
Random Forest	high	medium	likely	least likely
Extra Tree	high	fast	likely	least likely

Table 3.1: Advantages and Disadvantages of Linear Models, Gradient Boosting, Random Forest, and Extra Tree. These are the relative comparisons among the models and these comparisons do not consider other unmentioned models.

3.1.1 Linear Models

Linear Regression models and Logistic Regression models are very similar on how they train and predict on a dataset. Both models draw a best fit line through the training data. The best fit line aims to minimize errors from the line to the training data points. The only difference of the two model is Linear Regression model's outputs are continuous variables, while Logistic Regression model's outputs are categorical variables. Since the models generates a linear line, the models often underfit in complex dataset. However, for simple and small data, the models can perform well, while other complex models run into over-fitting problems.

3.1.2 Gradient Boosting Model

A Gradient Boosting model is a model that uses the concept of Boosting; however, instead of training different models, Gradient Boosting trains multiple decision trees. A decision tree [21] is a tree representation of decision the algorithm makes to reach to certain nodes

and leaves. Nodes hold combinations of the input features and leaves are the output corresponding to input features from the parent node. Gradient Boosting often yields good performance; however, it can generate overly complex trees, which overfits the data.

3.1.3 Random Forest Model

A Random Forest model utilizes multiple decision trees to make predictions. Random Forest, at its core, is very similar to Bagging. Random Forest splits data into multiple subset and grow each individual decision tree. Then, predictions of all decision trees are combined using one of the ensemble combination rules. Similar to Bagging, Random Forest helps to prevent over-fitting the model.

3.1.4 Extra Tree Model

An Extra Tree or Extremely Randomized Tree [9] model is very similar to Random Forest. The only difference is that the Extra Tree does not use the bootstrapping to create subsets from a training data. Instead, the subsets are taken randomly from the training data. This technique is proven to increase the predicting performance in some cases. In addition, it also reduces the computational time linked to the bootstrapping.

3.2 Training Components

In order to learn the pattern of a dataset, auto_ml goes through a training phase which includes multiple steps. Fig. 3.1 shows the overall structure of auto_ml and how each component connects and communicates across the framework. The following sections describe each of the components in this diagram.

3.2.1 Training Data

Inputs for auto_ml are a dataset, features description, and type of the dataset. The dataset is a data frame that includes input columns or features and an output column associated with those features. Each row of the dataset represents a set of features and an output. Features description specifies output's column, which columns to ignore, and which type, such as category, and date, of each feature.

3.2.2 Data Preprocessing

Data processing prepares the data for training the model. First, Data Cleaning removes duplicated rows, and any rows with missing features or unknown feature types. This eliminates the need for user involvement to check the for bad data. This process also splits the dataset into feature set and output set.

Then, the feature set is transformed to machine learning model readable inputs in Data Transformation process. This step transforms all columns of the feature set concurrently using multiple processes. This step utilizes the feature description to determine the appropriate transformation for each column of data. If the column contains continuous data, the data will be converted to floats. Otherwise, the step uses the Label Encoder[19] to convert categorical data to floats. Label Encoder visit all data in one column first and record , then revisit each data to convert it to a number.

Lastly, the number of features is reduced in Feature Selection. The goal of this step is to eliminate features that are not corresponding well with the output set. In this step, a Random Forest model is used to fit the feature set and output set. Then, the model produces the list of feature importances, which is a list of floats from 0 to 1, which indicates the importance of each feature. The value closer to 1 is more important. The method used to decide the

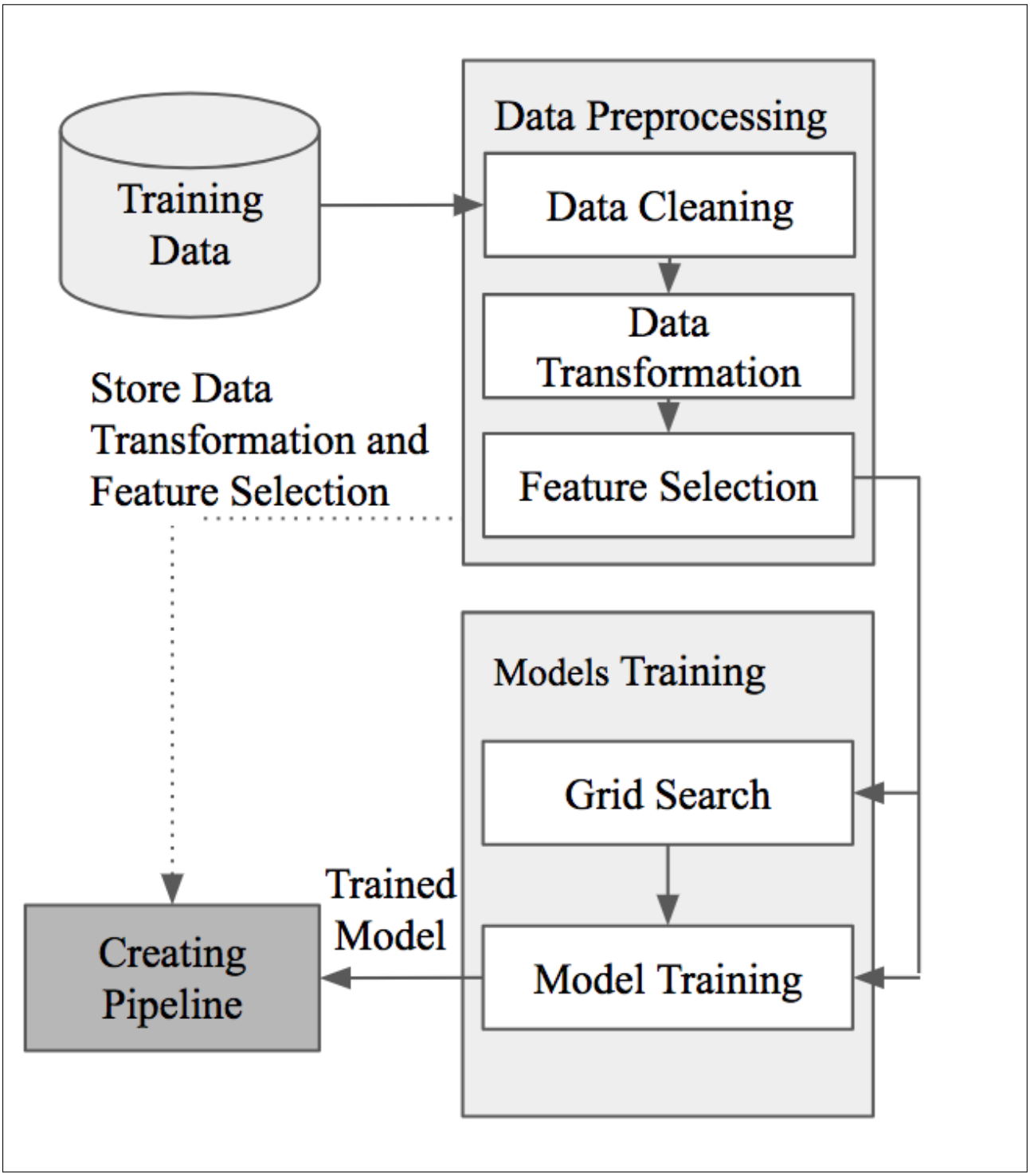

Figure 3.1: auto_ml's training process without Stacking. Each gray box is a component for the overall process. Each white box is a process runs within each component. Training Data is the user input. Solid arrows indicate main data flow and dashed arrows indicate secondary data flow.

important features is to pick any features that has its feature importance of at least 1/100th of the maximum feature importance.

3.2.3 Models Training

The Models Training component trains and selects the best model for the feature set and output set from previous component. The main process, grid search [2], looks through a list of models and decides which model best fits the data. Instead of searching for an optimal set of parameter for a model, grid search exhaustively uses cross-validation on each model to estimate how well each model performs on the given data. In other words, the set of models are the parameter space that grid search want to optimize. This also utilizes a pipeline, which will be discussed in the next section, to allow grid search validating multiple models on consistent settings.

After Grid Search finds the best model, it produces the trained model that best fits the data; however, the parameters for the model is not optimized.

3.2.4 Creating Pipeline

The Creating Pipeline component's goal is to simplify the predicting process with a use of Pipeline. Pipeline is a built in tool from scikit-learn library. The pipeline assembles the steps and their order in Training phase and then reuse it in the Model Training process and the Predicting phase. For auto_ml, this component stores the Data Transformation, Feature Selection, and the Trained Model. This ensures data processing consistency in Predicting phase. This component does not occur at the end of the training phase; but rather happens across the duration of the Training phase. Specifically, at the end of each step mentioned above, the step and their specific parameters are saved in Pipeline.

3.3 Prediction Components

The prediction process uses the Pipeline defined by the training process to evaluate data that it has not seen before. For regression types it produces a number and for classification types it predicts the class that the data set is a part of. The overall process is shown in Fig. 3.2. First, the Testing Data is processed with the same Data Transformation and Feature Selection steps as the training process. After the transformed data are obtained, Pipeline uses the trained model to get an output set for the Testing Data and return it to the developer.

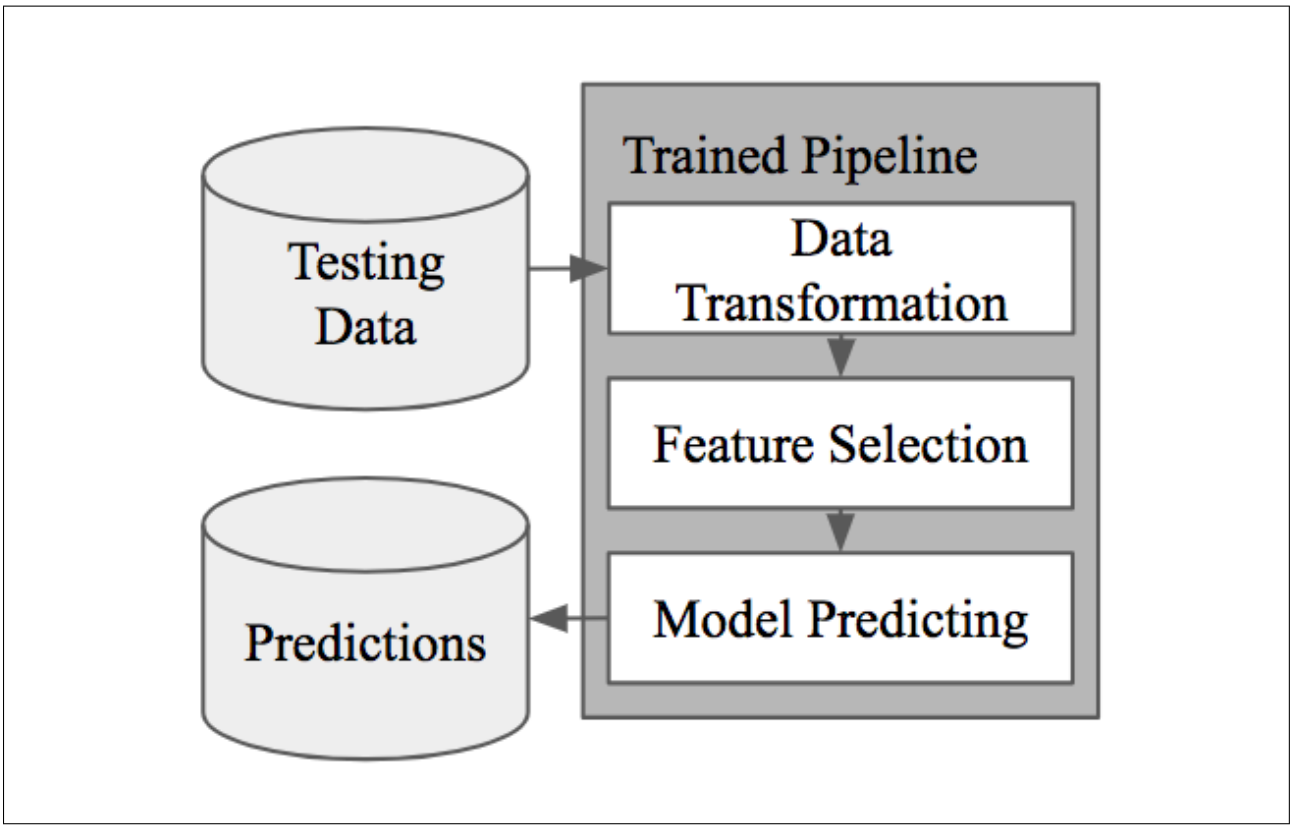

Figure 3.2: auto_ml's predicting process without Stacking. Each gray box is a component for the overall process. Each white box is a process runs within each component. Training Data is the user input. Predictions are the output of the framework. Solid arrows indicate main data flow.

Chapter 4

Stacking Implementation

While the auto_ml framework works well for many datasets, one of its weaknesses is overfitting when training on small data sets. This chapter describes the main contribution of this thesis: incorporating the Stacking ensemble technique in the auto_ml framework.

4.1 Machine Learning Models

The stacked auto_ml still uses the default models to utilize Stacking ensemble. As mentioned earlier in Section 3.1, these models have different advantages and disadvantages listed in Table 3.1. Using Stacking ensemble enables the framework to favor certain models for a specific dataset. For example, for a lower dimensional and simple dataset, Stacking ensemble learns to favor prediction from Linear models more than other models. The next section discusses about how the framework can achieve this goal.

4.2 Training Components

Adding stacking to the auto_ml framework causes dramatic changes to the training process. Fig. 4.1 shows the new overall training process including Stacking. This section identifies the modifications to the original system.

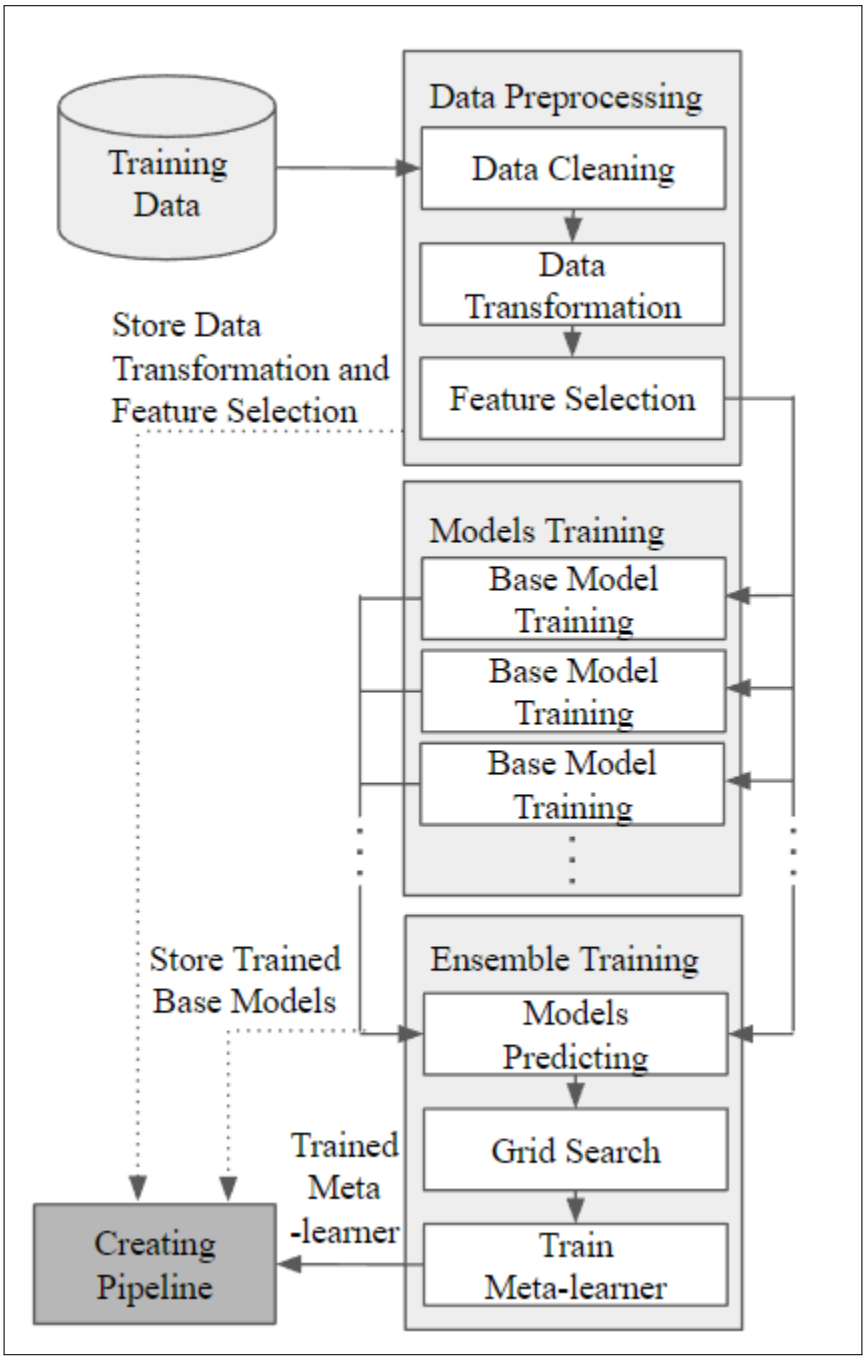

Figure 4.1: Stacked auto_ml's training process. Each gray box is a component for the overall process. Each white box is a process runs within each component. Training Data is the user input. Solid arrows indicate main data flow and dashed arrows indicate secondary data flow.

4.2.1 Training Data and Data Preprocessing

These processes are the identical processes explained in the Section 3.2.2.

4.2.2 Models Training

The Models Training component no longer uses Grid Search to find an optimal model to train. Instead, a process trains all the base models given from the framework. Due to the absent of Grid Search, this process can be executed concurrently because each model training process is independent from each other. Therefore, it can reduce the training time significantly compared to sequential execution. All models are trained using the same inputs from Data Preprocessing. Random sampling is not implemented for creating data subsets to train the models, because the thesis focuses on small datasets and it would lower the performance significantly if the models aren't well trained with enough data. Finally, trained models are used in Ensemble Training and Creating Pipeline.

4.2.3 Ensemble Training

The Ensemble Training trains the meta-learner. Algorithm 1 shows the steps used to create inputs for meta-learner and train the meta-learner. First, each trained base model uses the data to get a set of predictions. The process uses same data used in Models Training for the similar reasons In the next section, the performance of using the whole data for training is compared to different splitting ratio methods to confirm that using the whole data has a better performance. The predictions from all models are combined and used to train a Linear Regression model. This implementation uses Linear Regression model as the meta-learner because predictions from different base models are very close to each other and there are

very little to no outliers, which would highly affect the performance of Linear Regression.

The Linear Regression is trained using grid search with a list of possible parameters in the model, the predictions, and outputs of the original data. Grid search is being used differently in this process, it does not search for the best model among multiple models, but rather search for the best parameter set for Linear Regression model. Grid search also trains the model with the best possible parameter set.

Algorithm 1 Algorithm for training meta-learner

Input: Training data, (X, Y). List of trained base models: M.
Output: Trained meta-learner, P

```
Initialize x̂
for m in M do
    p = m.predict(X)
    x̂.append(p)
end for
params = gridsearch parameters for Linear Regression model
results = gridserach.fit(x̂, Y, params)
P = results.best_model
return P
```

4.2.4 Creating Pipeline

This component is very similar to the original auto_ml with an slight modification. However, Pipeline also stores all the trained base models for Predicting phase.

4.3 Predicting Components

Similar to the original auto_ml, Testing Data undergoes two transformation processes: Data Transformation and Feature Selection. Then, each base model will make a set of intermediate

predictions using the transformed data. These intermediate predictions are combined and used to get a final prediction from the trained meta-learner.

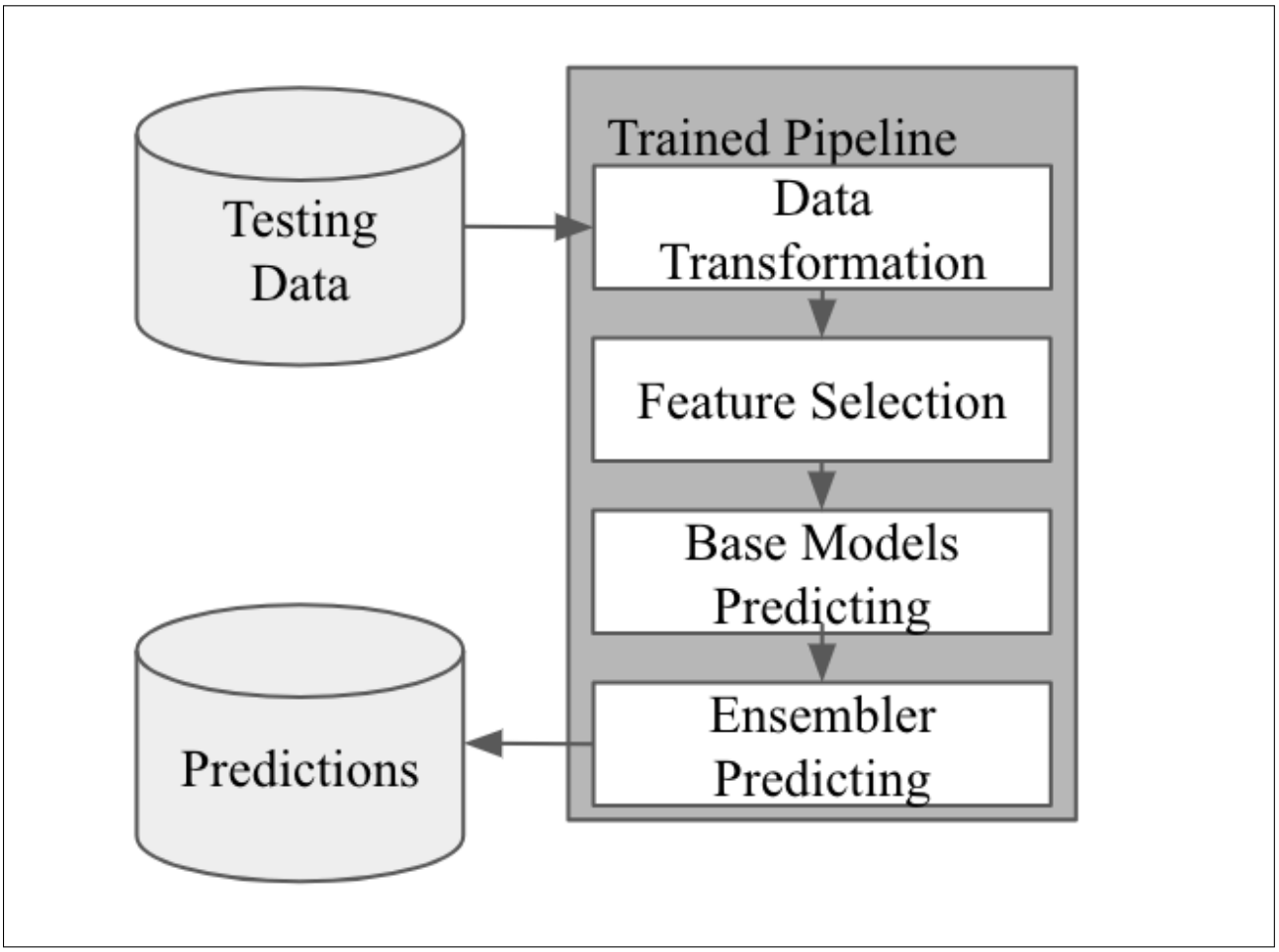

Figure 4.2: Stacked auto_ml's predicting process. Each gray box is a component for the overall process. Each white box is a process runs within each component. Training Data is the user input. Solid arrows indicate main data flow.

Chapter 5

Results

Several sets of results are presented to compare auto_ml, Stacked auto_ml , and MLP trained machine learning algorithms to determine how each perform on small data sets, which are susceptible to over-fitting. First the original auto_ml algorithm is compared with an MLP neural network for baseline statistics without stacking. The auto_ml framework is also compared with auto-sklearn, which is a competing framework. Finally, a comparison between auto_ml and auto_ml with stacking is provided to show the enhanced robustness to over-fitting.

All tests are run on a machine with an 8 core AMD Ryzen 1700 processor, 16GB of ram, and M.2 SATA storage with a 256GB hard drive. Datasets used in this section are from UCI Machine Learning Repository [14] and the Department of Statistics at the University of Florida [25]. There are a total of 50 datasets used for these tests. They're relatively small datasets, which have less than 500 rows (i.e. data1 has 104 rows, data2 has 153 rows, and data3 has 249 rows). All datasets are split into training sets and testing sets with the ratio of 7:3. Table 5.1 shows the description of each dataset used for testing.

After training each individual model or framework, an R-squared (R^2) score is used to measure how close the predictions of a model or a framework is to the testing set. R^2 is also known as the coefficient of determination; it is interpreted as the proportion of the variance in the dependent variable that is predictable from the independent variable [16]. This thesis uses an implementation of R^2 from the scikit-learn library. The implementation ranges the

R^2 score from 1 to any finite negative number, because the model can be arbitrarily worse. To ensure the performance consistency of the model or framework, each test is repeated five times after the training and testing sets are obtained.

Dataset	Number of Rows	Number of Features	Area
data1	258	13	E-commerce
data2	249	19	Gaming
data3	2448	12	Gaming
data4	104	10	School
data5	153	7	Winery
data6	324	32	E-commerce
data7	44	6	Politics
data8	72	14	Politics
data9	499	11	School
data10	95	5	E-commerce
data11	157	3	Urban
data12	8905	7	Urban
data13	176	3	Politics
data14	150	4	Medical
data15	118	4	Medical
data16	509	8	School
data17	230	6	Winery
data18	506	14	E-commerce
data19	342	22	Gaming
data20	395	10	E-commerce
data21	425	16	Urban
data22	341	18	Winery
data23	394	10	Gaming
data24	417	22	E-commerce
data25	439	22	School
data26	476	16	School
data27	283	18	School
data28	395	16	Politics
data29	429	10	Politics
data30	494	10	Politics
data31	361	18	Space
data32	389	22	Space
data33	394	18	Medical
data34	492	20	Medical
data35	454	10	Medical
data36	403	20	Computer
data37	404	16	Computer
data38	495	16	Mechanical
data39	422	129	Mechanical
data40	439	10	School
data41	393	10	Space
data42	442	11	Politics
data43	359	10	E-commerce
data44	402	16	Medical
data45	359	10	School
data46	452	10	Mechanical
data47	365	16	Agriculture
data48	450	10	Mechanical
data49	404	6	Medical
data50	450	10	Gaming

Table 5.1: Descriptions of each dataset. The number of rows indicates how many data points there are in the dataset. The number of features indicates how many different inputs and outputs there are. The area shows the field that a dataset is collected for.

5.1 auto_ml vs MLP

This test is conducted to compare the performance of auto_ml and MLP. Fig. 5.1 shows the performance across the first 17 datasets. Except for data15, auto_ml performs better than MLP. This is expected since MLP usually requires a large dataset in order to perform well. Fig. 5.2 shows that auto_ml does require more time to go through the training process. Although, there are considerable differences in runtime, the runtime for auto_ml are still relatively small considering its performance. This difference is due to the models training step in auto_ml. As mention earlier, auto_ml selects an optimal model out of the four default model and use it to predict a given dataset. Table 5.2 shows which model is selected, by auto_ml, for all 50 datasets. The framework chooses a Gradient Boosting Regression model for 26% of the datasets, a Extra Trees Regression model for 36% of the datasets, a Random Forest Regression model for 10% of the datasets, and a Linear Regression model for 28% of the datasets.

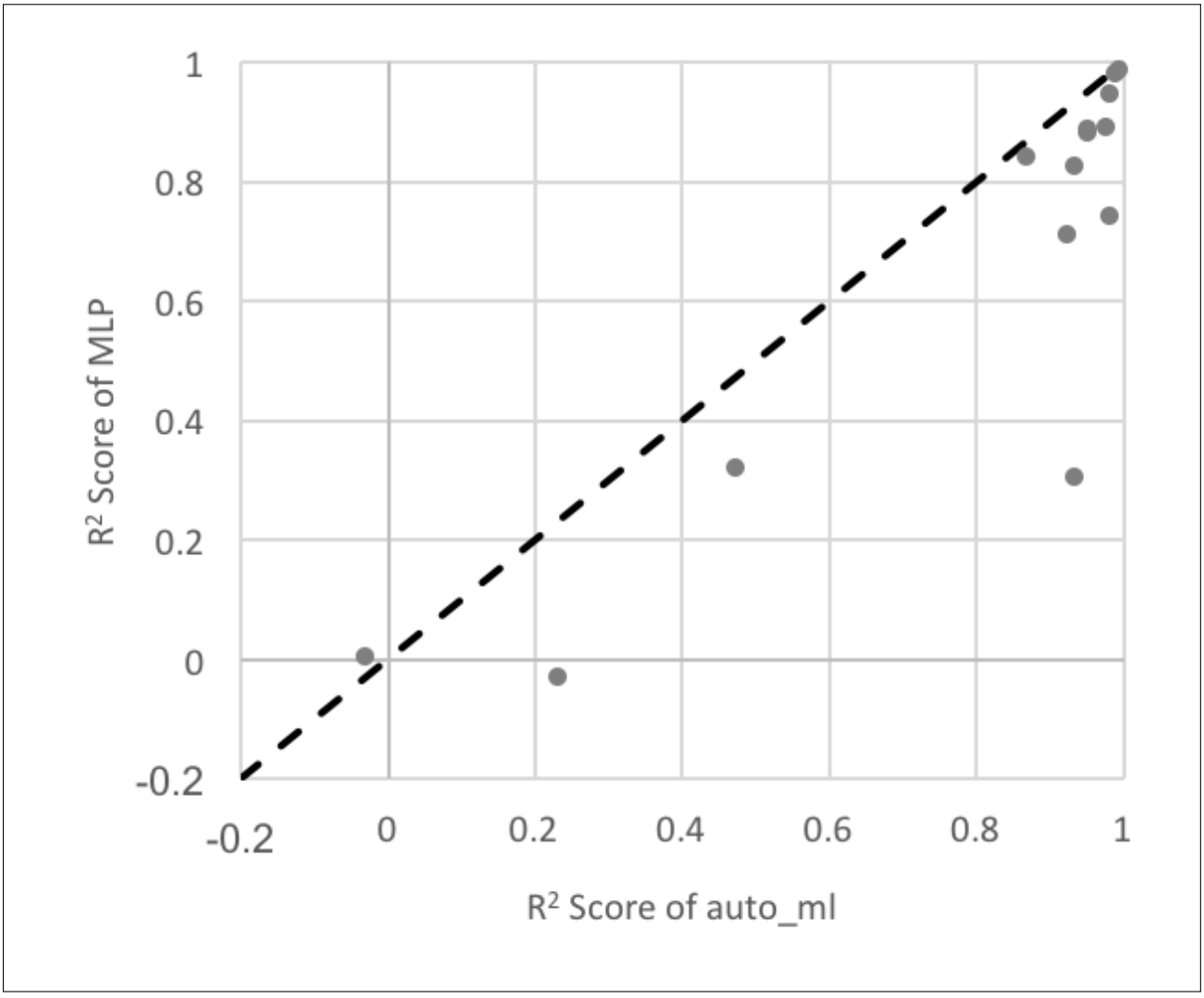

Figure 5.1: R^2 Scores of auto_ml and MLP. Each dot represents a dataset. If a dot is below the dashed line, it means auto_ml has a better score in the dataset. The R^2 scores for MLP of data4, data8, and data12 are not shown because the MLP scores are very negative and beyond the scope of the graph.

Dataset	Selected Model
data1	Gradient Boosting Regression
data2	Extra Trees Regression
data3	Gradient Boosting Regression
data4	Extra Trees Regression
data5	Extra Trees Regression
data6	Linear Regression
data7	Gradient Boosting Regression
data8	Random Forest Regression
data9	Random Forest Regression
data10	Extra Trees Regression
data11	Gradient Boosting Regression
data12	Gradient Boosting Regression
data13	Extra Trees Regression
data14	Linear Regression
data15	Gradient Boosting Regression
data16	Gradient Boosting Regression
data17	Extra Trees Regression
data18	Extra Trees Regression
data19	Extra Trees Regression
data20	Linear Regression
data21	Extra Trees Regression
data22	Linear Regression
data23	Linear Regression
data24	Extra Trees Regression
data25	Extra Trees Regression
data26	Extra Trees Regression
data27	Linear Regression
data28	Extra Trees Regression
data29	Linear Regression
data30	Random Forest Regression
data31	Linear Regression
data32	Extra Trees Regression
data33	Linear Regression
data34	Gradient Boosting Regression
data35	Gradient Boosting Regression
data36	Gradient Boosting Regression
data37	Extra Trees Regression
data38	Extra Trees Regression
data39	Gradient Boosting Regression
data40	Linear Regression
data41	Linear Regression
data42	Gradient Boosting Regression
data43	Linear Regression
data44	Extra Trees Regression
data45	Linear Regression
data46	Gradient Boosting Regression
data47	Extra Trees Regression
data48	Linear Regression
data49	Extra Trees Regression
data50	Linear Regression

Table 5.2: This table shows the selected model for each dataset from auto_ml.

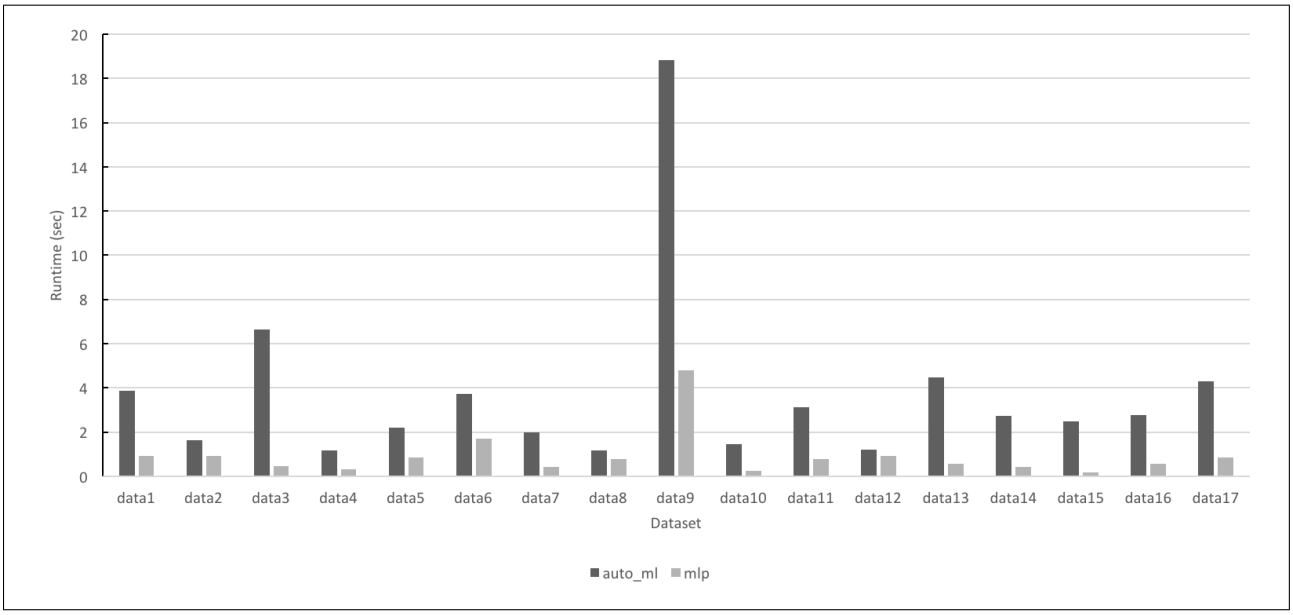

Figure 5.2: Training runtime of auto_ml and MLP. auto_ml runtime is within an order of magnitude of MLP runtime

5.2 auto_ml vs auto-sklearn

This test is used to verify how auto_ml's performance is compared to auto-sklearn's. The default settings are indented used for the two frameworks to compare and select their best model to predict given datasets. However, by default, auto-sklearn trains its hyper-parameter optimization and other tools for an hour regardless the data size; this is too long for training small data. Therefore, this training time is reduced to 30 minutes. Fig. 5.3 shows that auto_ml's R^2 scores are very close to auto-sklearn's, although auto-sklearn's scores are slightly higher. The auto-sklearn framework uses a constant runtime, which is approximately 30 minutes, to train each dataset. On the other hand, auto_ml's runtimes are dependent to the data size; hence, it runs faster for small datasets. As shown in Fig. 5.4, there are big runtime differences.

To observe auto-sklearn's performance given a fast training time, the benchmarking code is modified to change the default value of 30 minutes to 10 seconds. Figs. 5.3 and 5.4 also includes the R^2 scores and runtime of the sped-up auto-sklearn. As shown in the figures, the performance of auto-sklearn drops considerably when the training time is shorten. In addition, training time for auto-sklearn can not be any arbitrary low number because a component of auto-sklearn, SMAC, requires a fix runtime based on the data size. To speed up auto-sklearn, it requires expert's intervention to search for an optimal training time.

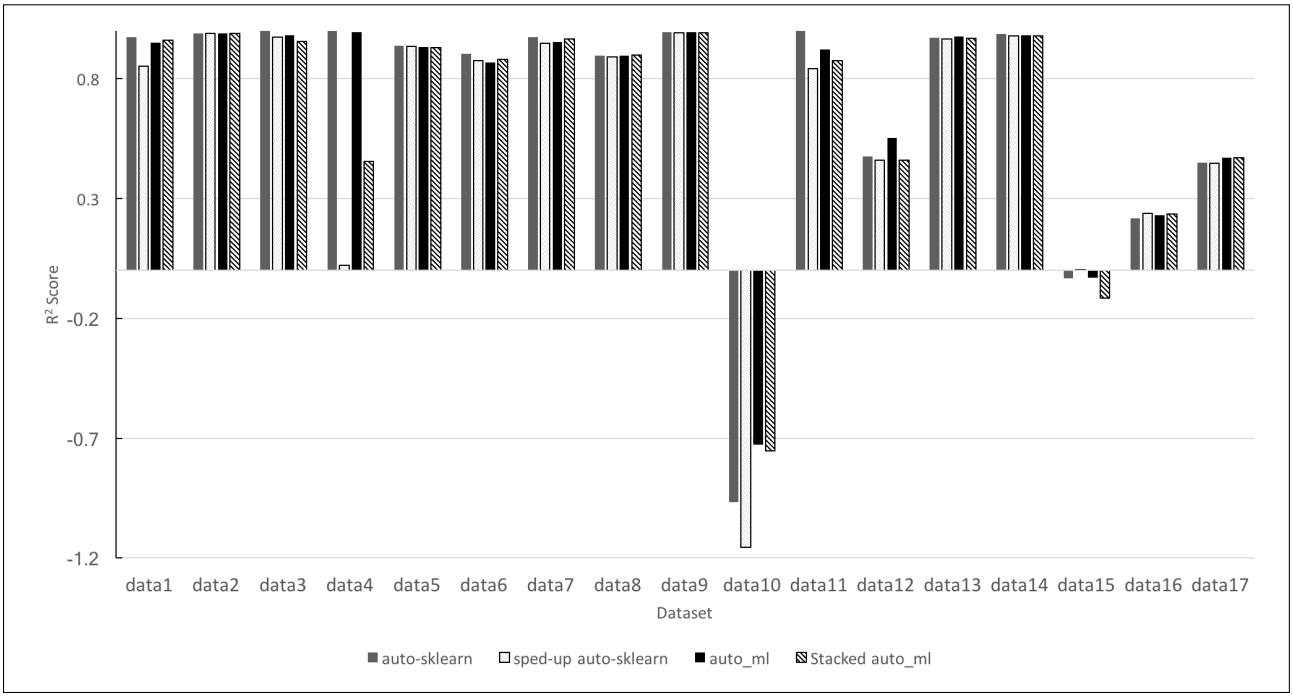

Figure 5.3: R^2 Scores of auto_ml and auto-sklearn (Closer to 1 is better).

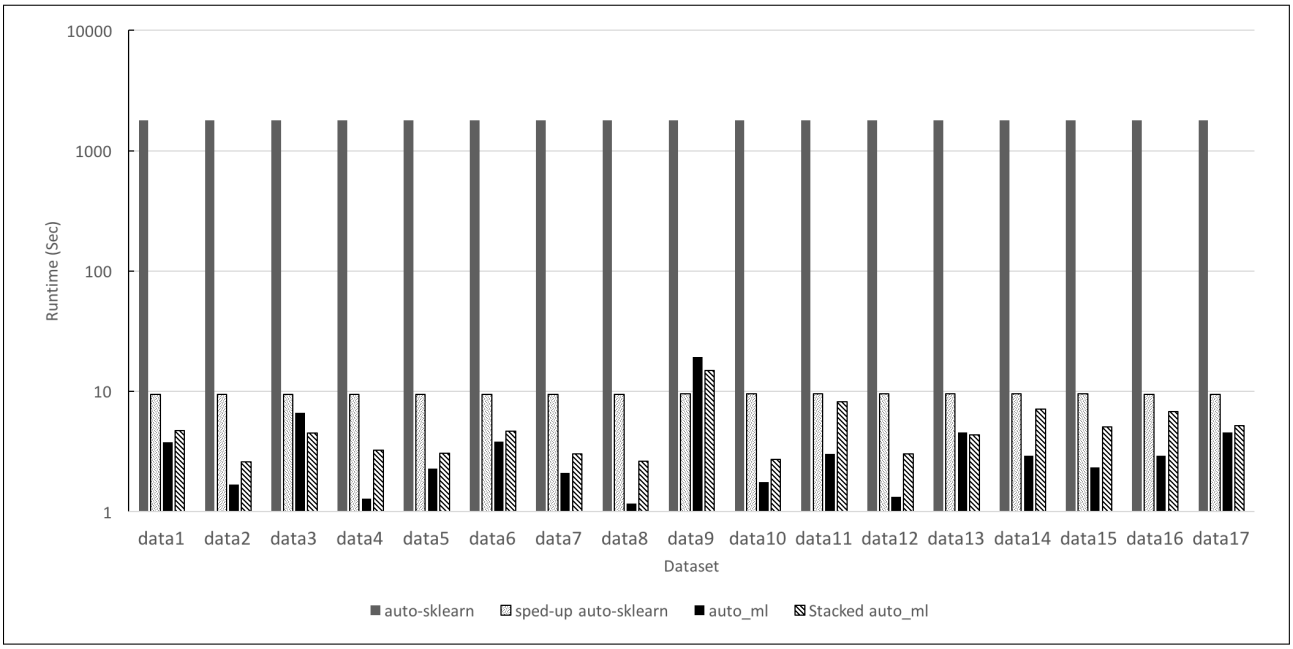

Figure 5.4: R^2 Training runtime of auto-sklearn, speed-up auto-sklearn, auto_ml and Stacked auto_ml. This figure shows the runtime of the test conducted to gather the results for Fig 5.3.

5.3 Stacked auto_ml vs auto_ml

This test uses all 50 datasets to compare the performance of Stacked auto_ml and auto_ml. For each dataset, different training space percentages, from 30% to 95%, are used to train the frameworks. The goal is to monitor the performance of each framework across different training sizes to observe how well the framework avoids over-fitting. Figs. 5.5 to 5.21 are the results for the first 17 datasets. Each figure consists of R^2 score for each different training space percentage from a dataset. Some figures show the performance of the two frameworks when the datasets are not susceptible to over-fitting; Fig. 5.6 shows their performance without over-fitting. There are also figures that show the performance of the frameworks when over-fitting occurs; Fig. 5.7 shows the frameworks' performance when they overfit the dataset. In list of figures, over-fitting occurs when there are a considerable drop as the training space percentage increases.

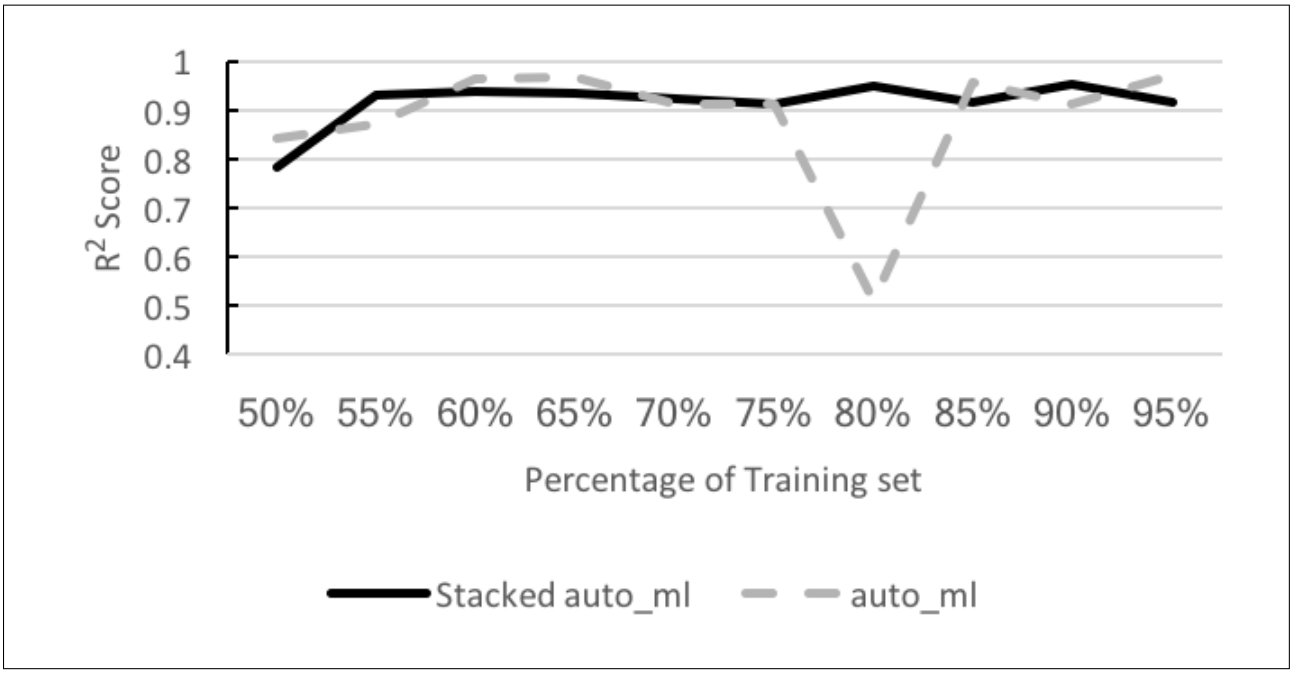

Figure 5.5: R^2 Scores of Stacked auto_ml and auto_ml for data1 (closer to 1 is better).

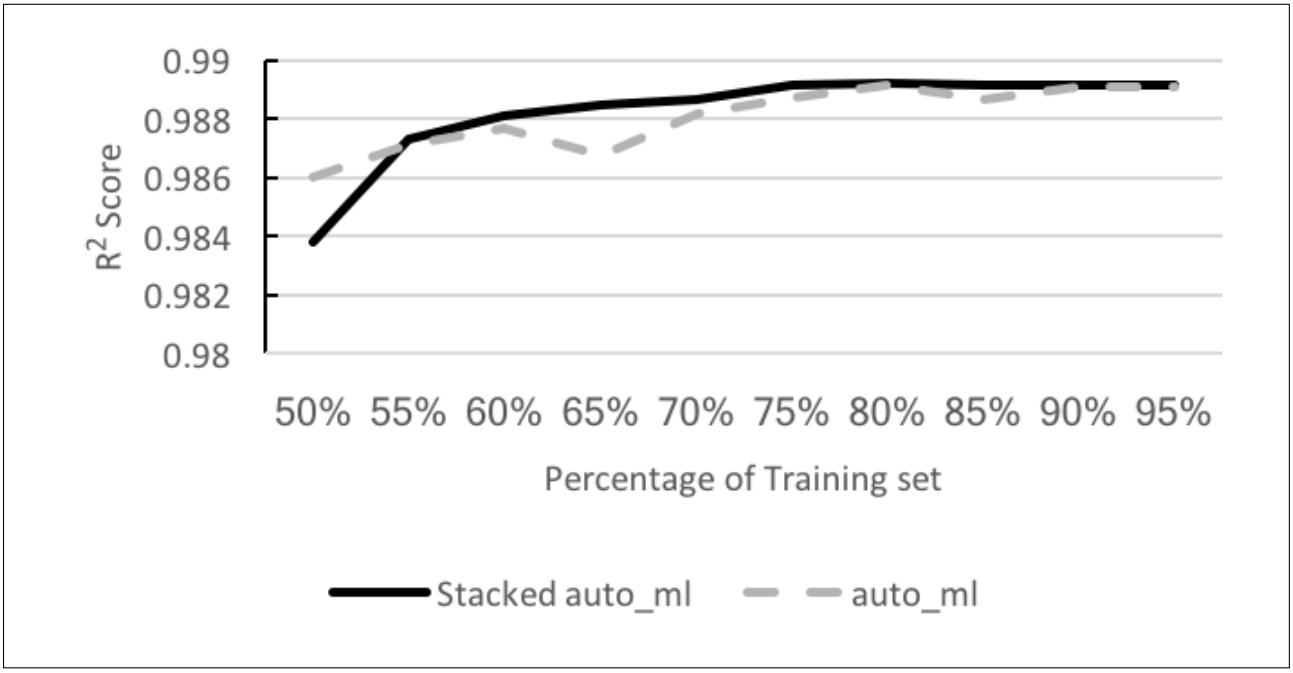

Figure 5.6: R^2 Scores of Stacked auto_ml and auto_ml for data2 (closer to 1 is better).

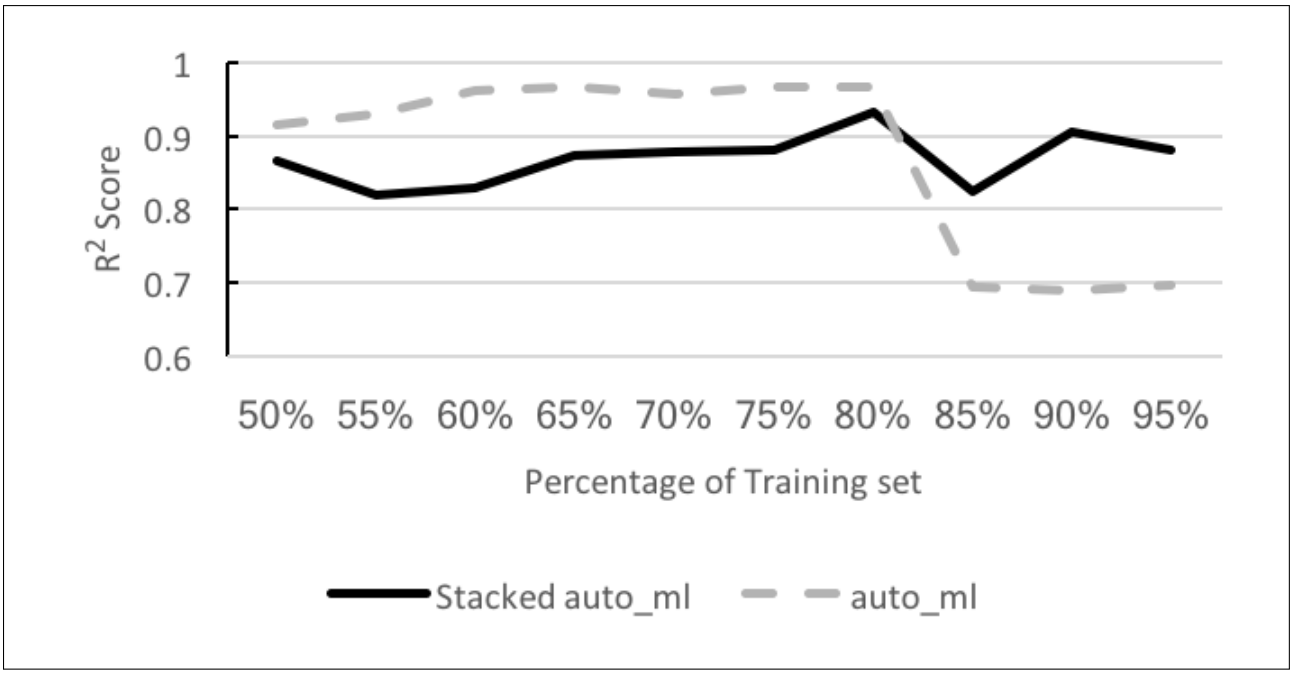

Figure 5.7: R^2 Scores of Stacked auto_ml and auto_ml for data3 (closer to 1 is better).

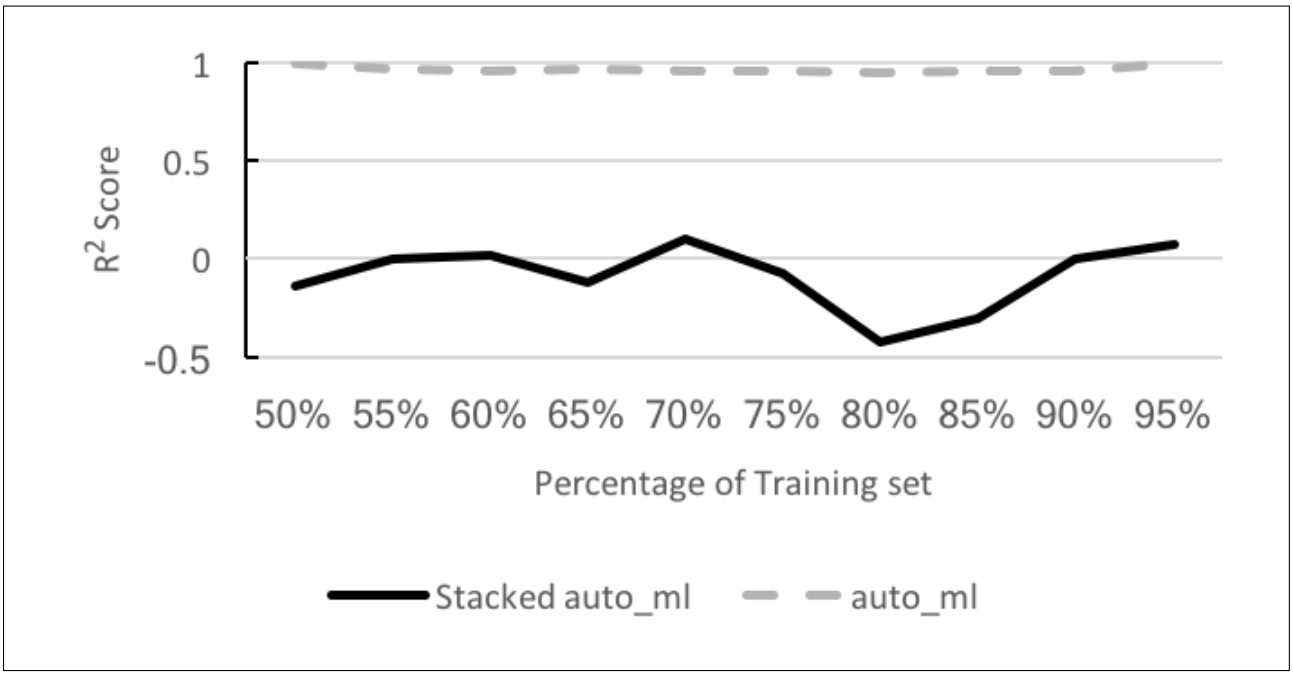

Figure 5.8: R^2 Scores of Stacked auto_ml and auto_ml for data4 (closer to 1 is better).

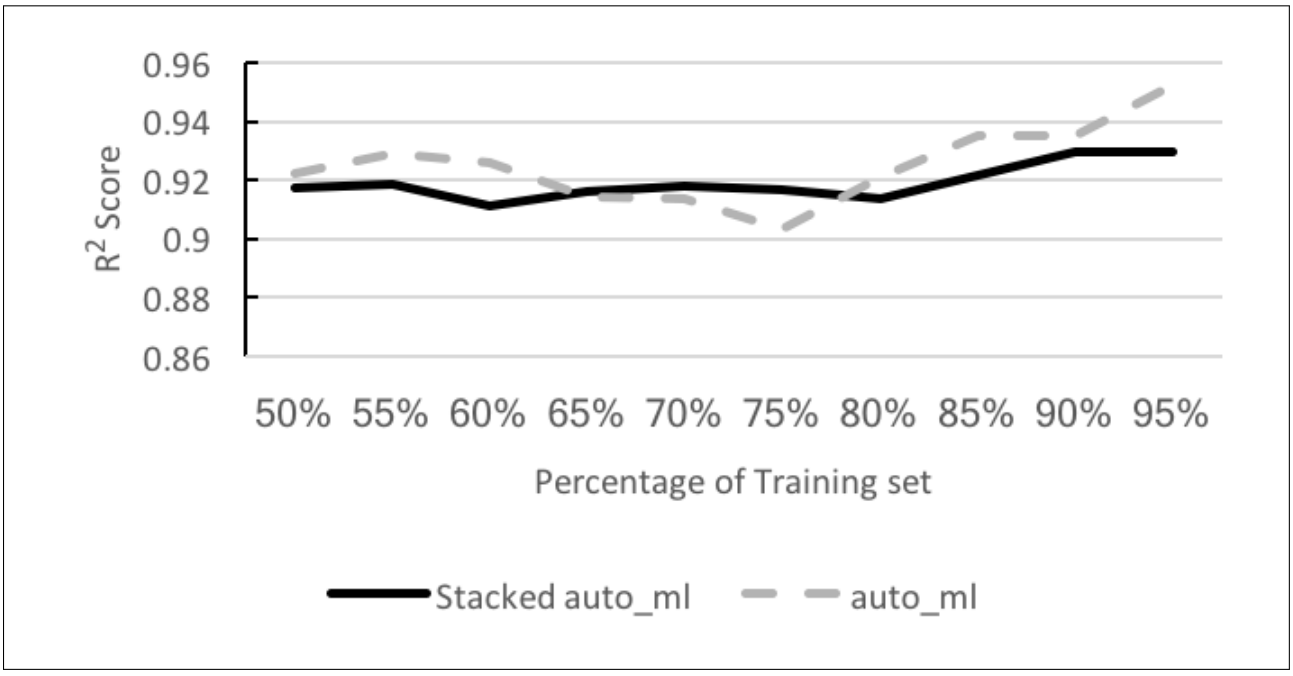

Figure 5.9: R^2 Scores of Stacked auto_ml and auto_ml for data5 (closer to 1 is better).

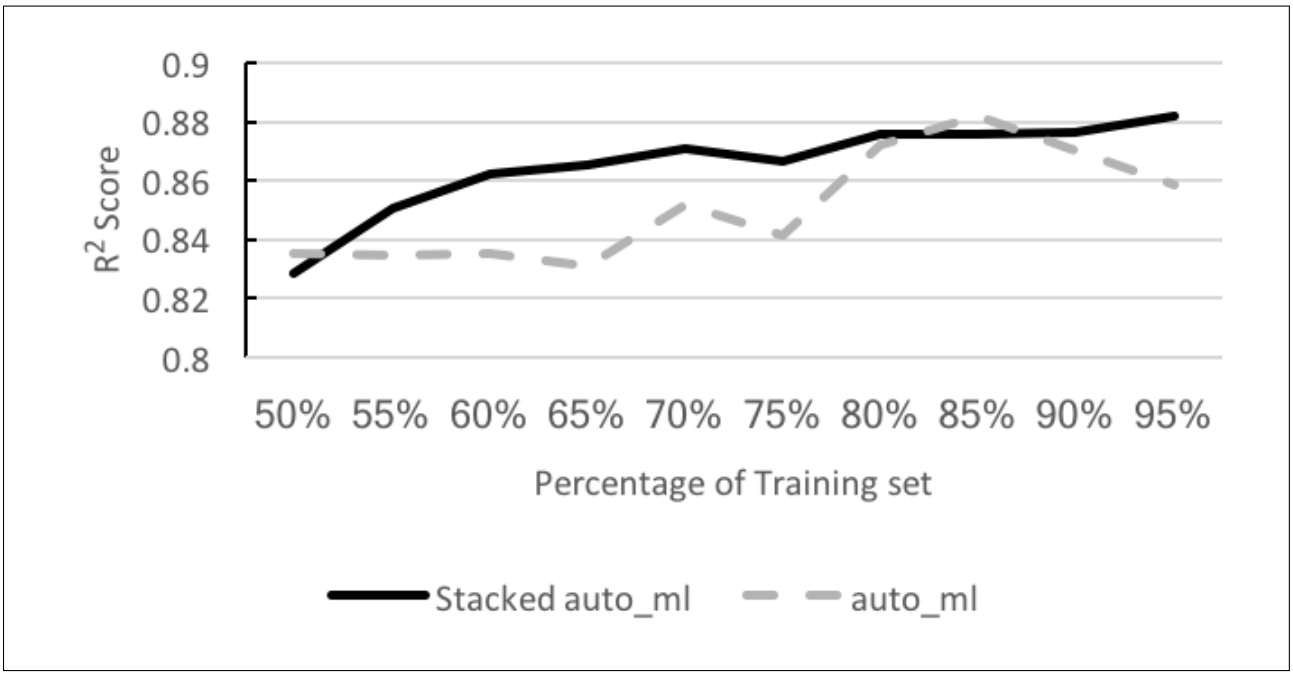

Figure 5.10: R^2 Scores of Stacked auto_ml and auto_ml for data6 (closer to 1 is better).

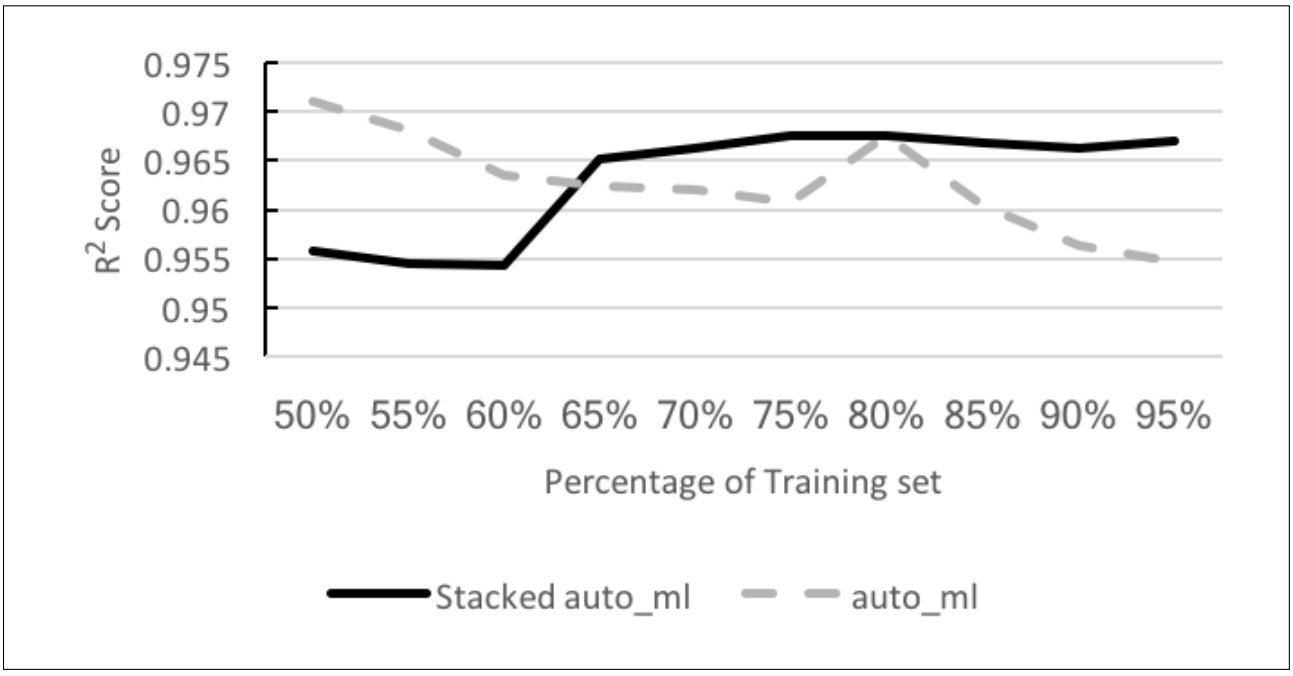

Figure 5.11: R^2 Scores of Stacked auto_ml and auto_ml for data7 (closer to 1 is better).

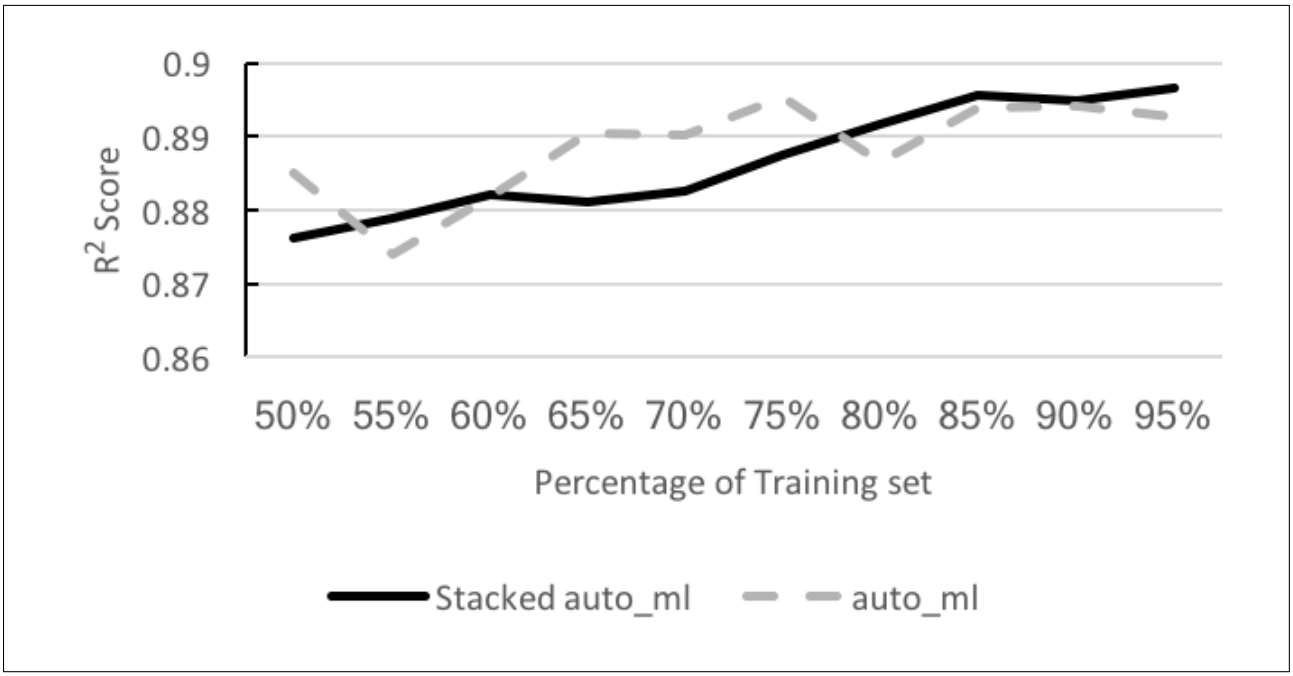

Figure 5.12: R^2 Scores of Stacked auto_ml and auto_ml for data8 (closer to 1 is better).

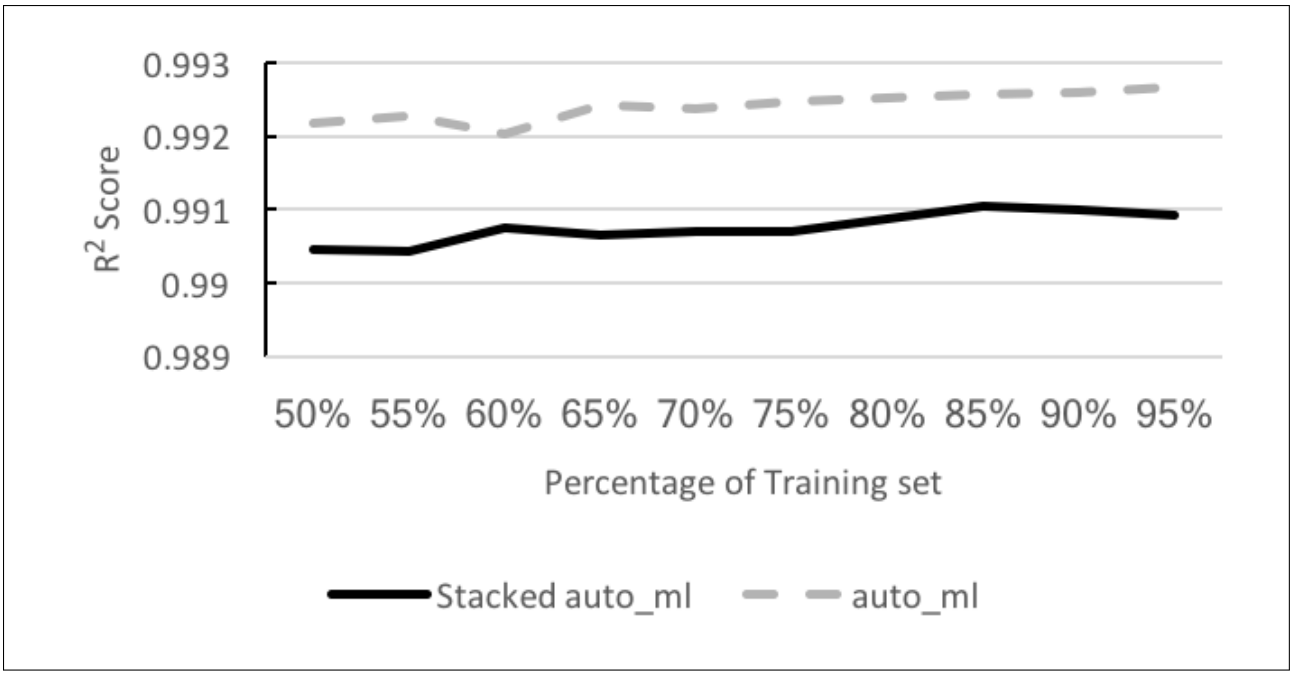

Figure 5.13: R^2 Scores of Stacked auto_ml and auto_ml for data9 (closer to 1 is better).

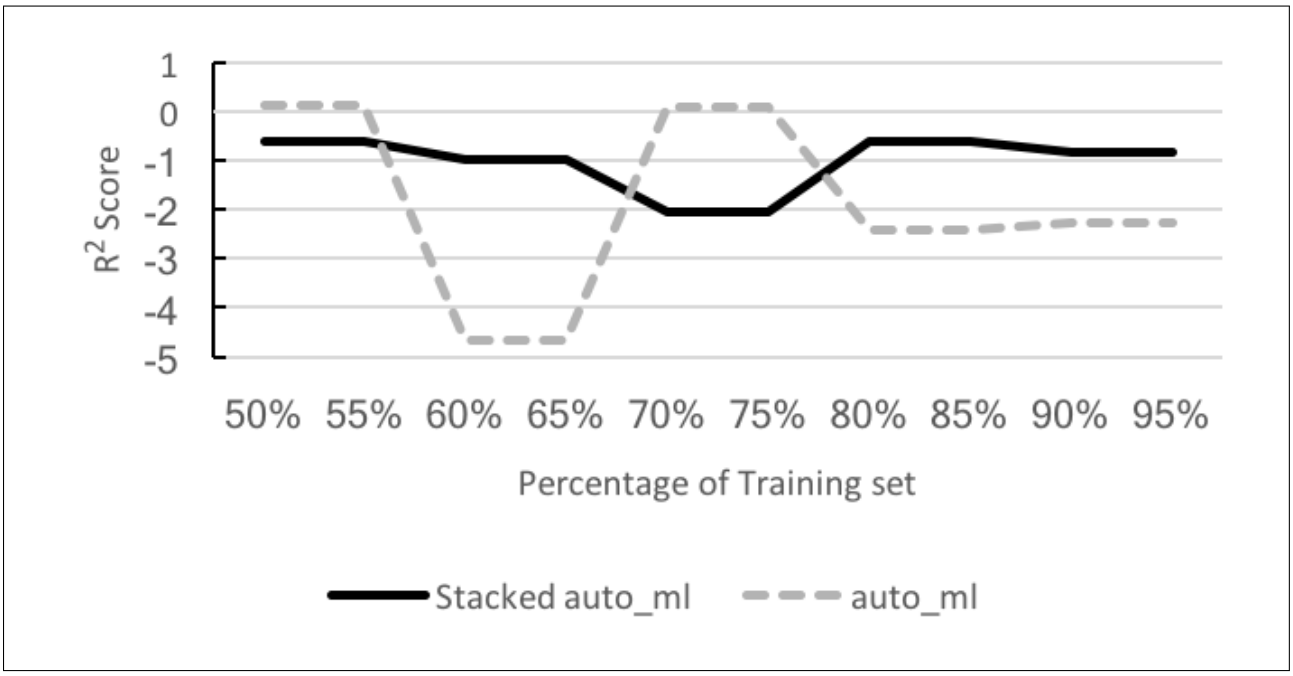

Figure 5.14: R^2 Scores of Stacked auto_ml and auto_ml for data10 (closer to 1 is better).

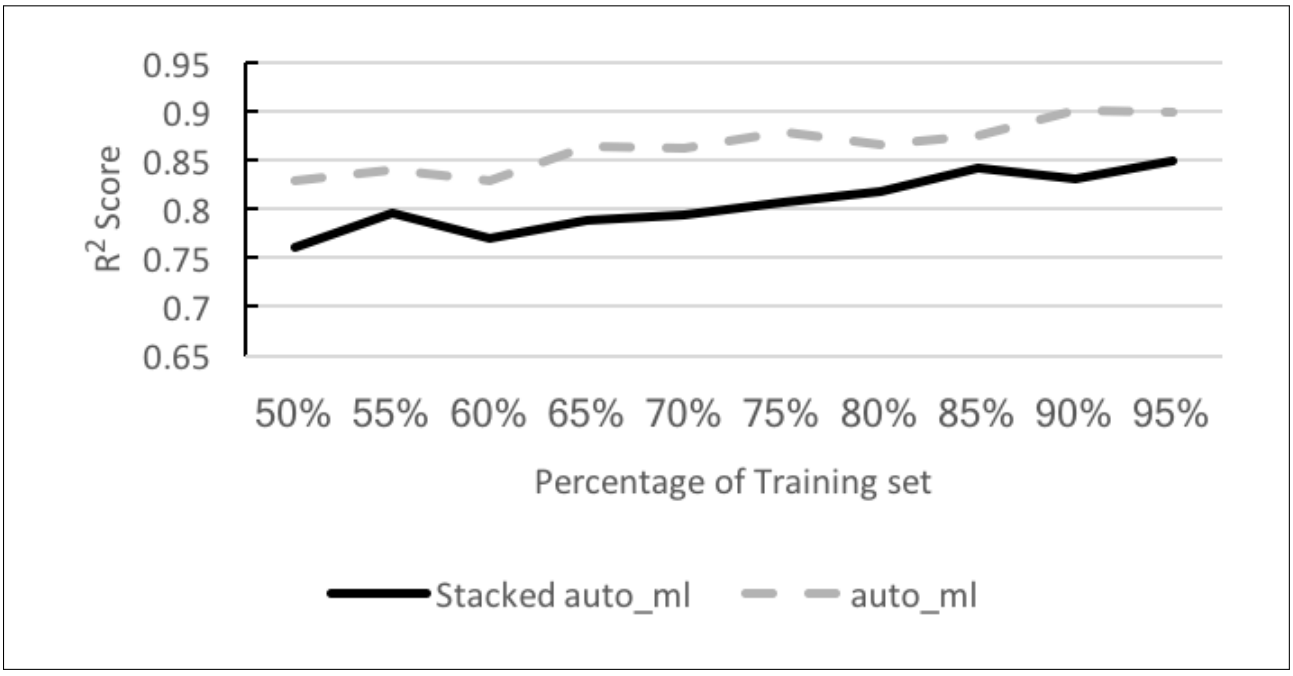

Figure 5.15: R^2 Scores of Stacked auto_ml and auto_ml for data11 (closer to 1 is better).

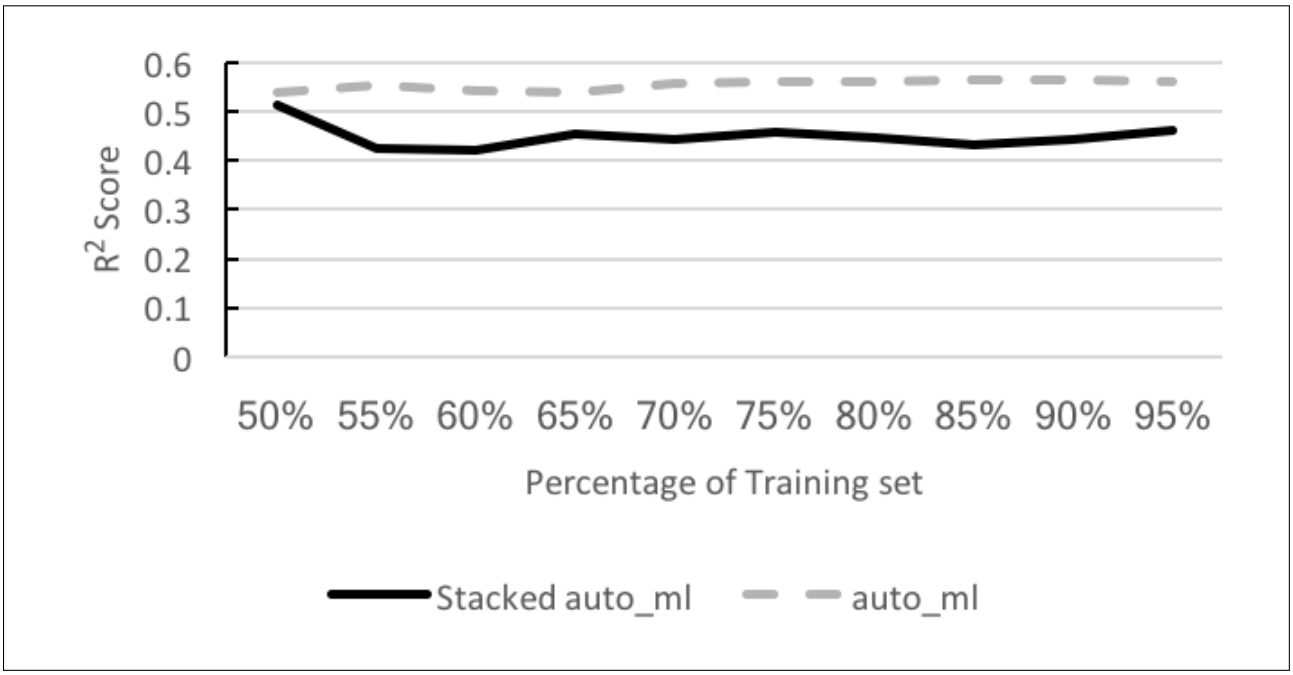

Figure 5.16: R^2 Scores of Stacked auto_ml and auto_ml for data12 (closer to 1 is better).

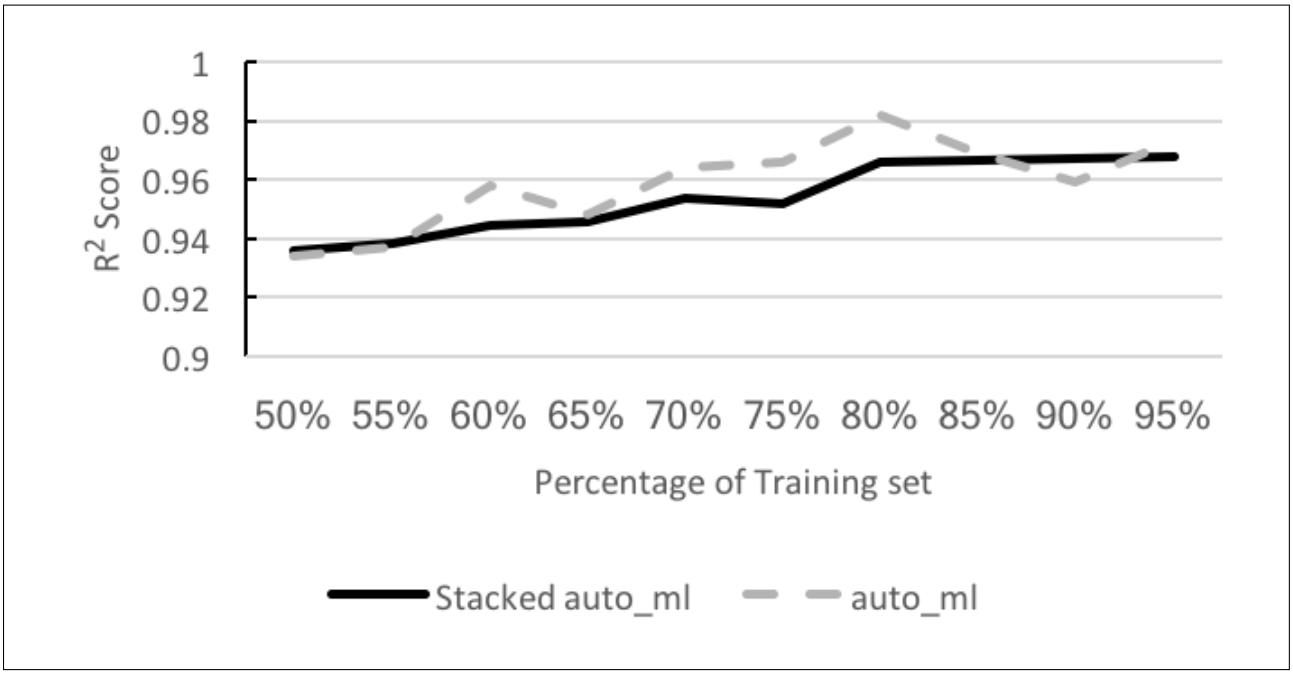

Figure 5.17: R^2 Scores of Stacked auto_ml and auto_ml for data13 (closer to 1 is better).

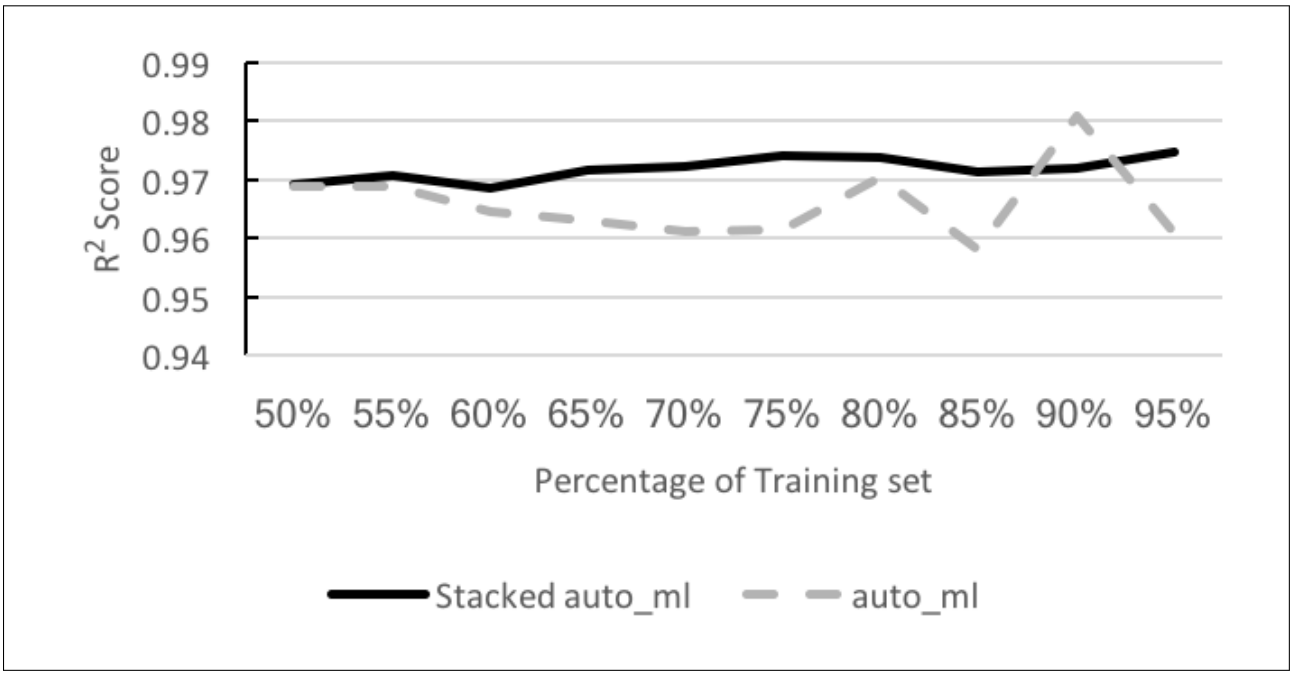

Figure 5.18: R^2 Scores of Stacked auto_ml and auto_ml for data14 (closer to 1 is better).

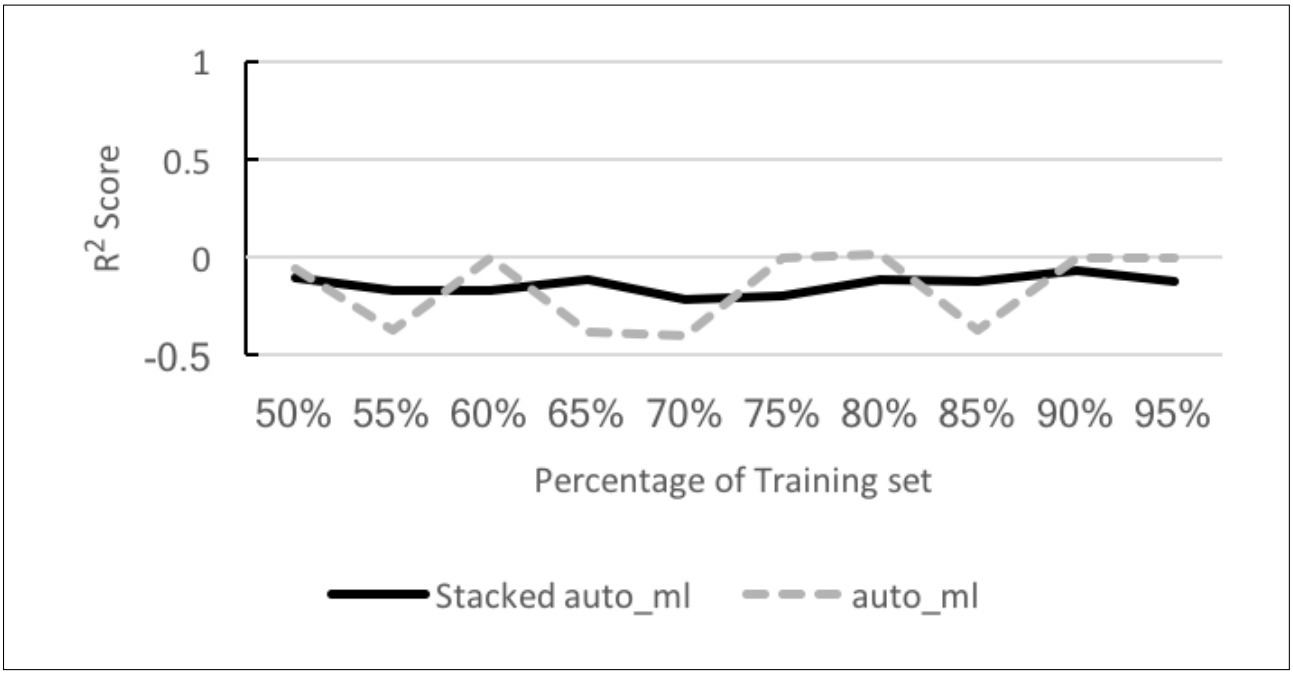

Figure 5.19: R^2 Scores of Stacked auto_ml and auto_ml for data15 (closer to 1 is better).

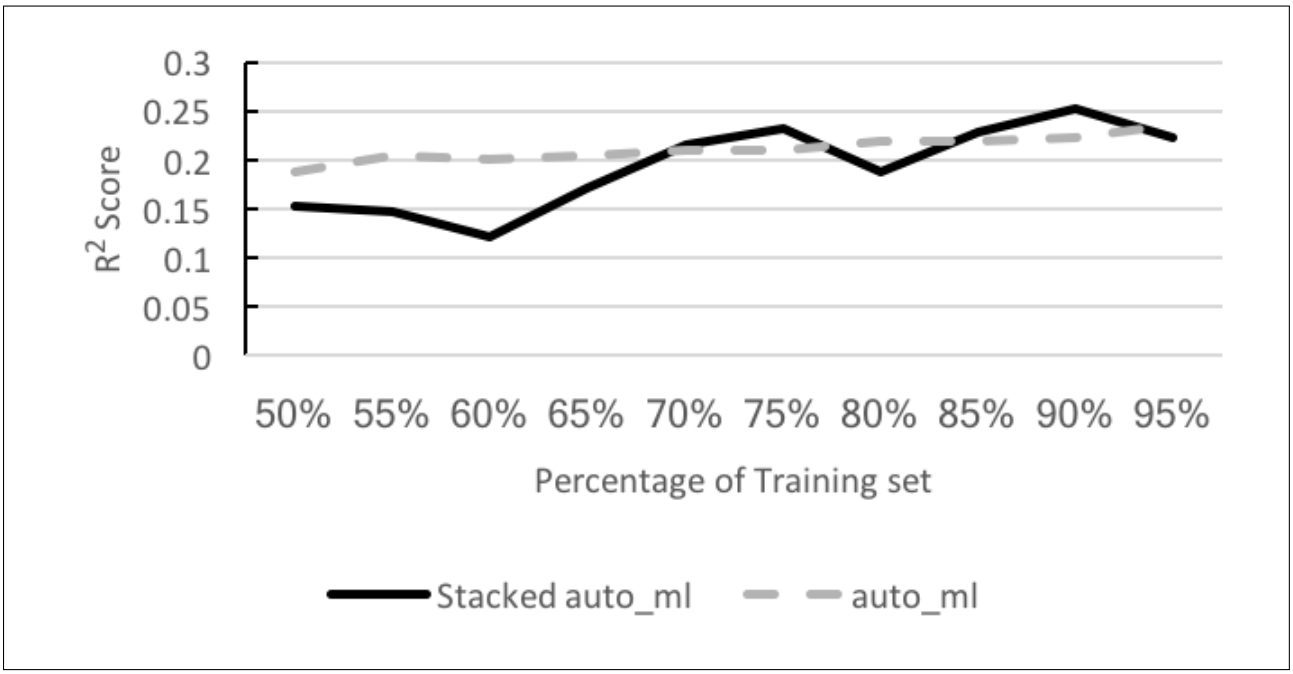

Figure 5.20: R^2 Scores of Stacked auto_ml and auto_ml for data16 (closer to 1 is better).

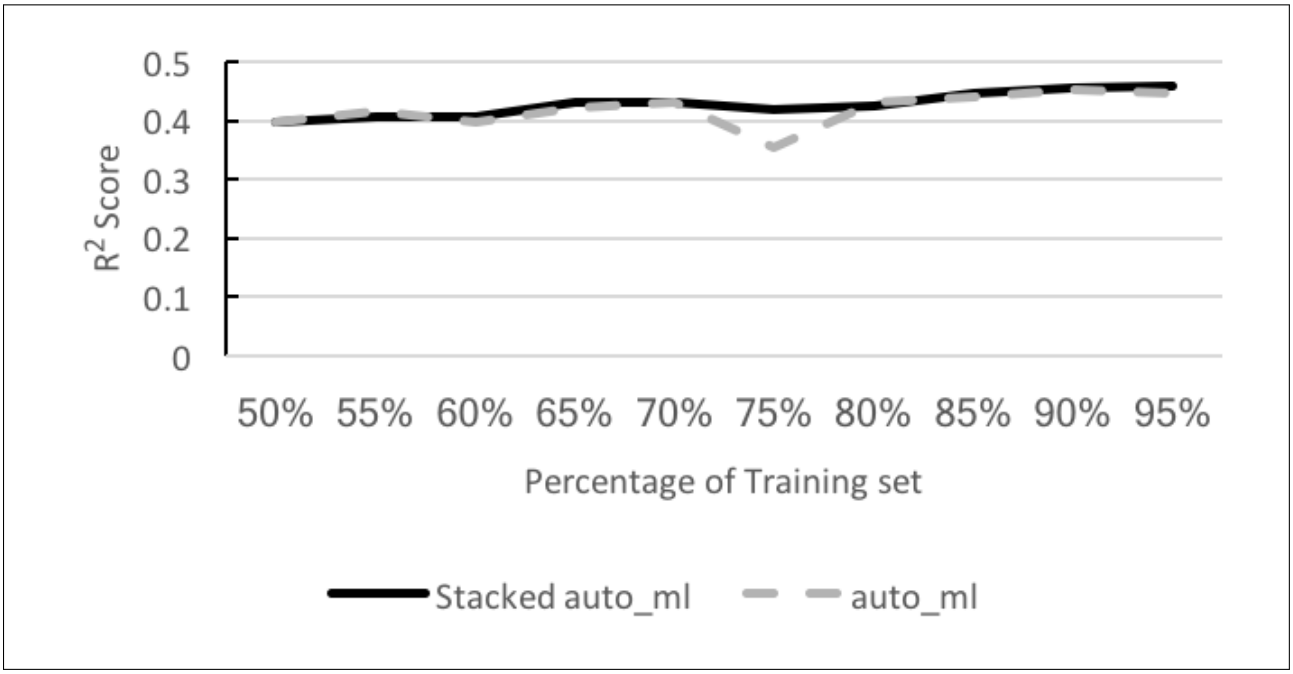

Figure 5.21: R^2 Scores of Stacked auto_ml and auto_ml for data17 (closer to 1 is better).

Table 5.3 compares how well each framework performs against over-fitting for the first 17 datasets. The table composes of Δ(Variance), Δ(Max-Min), Δ(Max Drop) of the frameworks across 17 datasets. A bold number indicates the score is better for Stacked auto_ml. Negative integers indicate that Stacked auto_ml scored less than auto_ml. Each column is calculated by computing the variance, maximum, minimum, and max drop of the frameworks for each dataset; then, calculate the differences. Max drop is calculated by taking the most negative change of the R^2 score of each framework in a dataset. The higher max drop dedicates the more vulnerability of a framework to over-fitting. For example, in Fig. 5.7, the drop for auto_ml from 80% to 85% is considered as the max drop of the framework for this dataset. This shows that the framework starts to over-fit the training set and the performance drops significantly. On the other hand, Stacked auto_ml shows a much smaller drop. Fig. 5.22 reveals the over-fitting problem from the test on data3 by getting predictions and calculating R^2 scores from using the whole training set and the testing set. When using training set and after 80%, the R^2 scores increase as more training data are used to train the framework. On the other hand, when using testing set, the R^2 scores drop significantly after 80%. Fig. 5.23 shows that when the framework does not encounter over-fitting in data2, the two curves should be very close and increase as more training data are used.

In terms of Δ(Variance), there are 9 datasets have a better Δ(Variance) for Stacked auto_ml, and 8 datasets do not. This is due to the fact the the variance includes changes, negative or positive, of the R^2 score across multiple training space. That's why Δ(Variance) can not be used alone to justify the over-fitting prevention of the framework. That is why Δ(Max-Min) and Δ(Max Drop) are included. Δ(Max-Min) columns shows that the majority of datasets have smaller gap between the maximum and minimum R^2 score for Stacking auto_ml. This means that the Stacked auto_ml is more consistent across multiple training size; hence, it is more robust against over-fitting. However, Δ(Max-Min) column is not consistent with

Δ(Max Drop) columns in term of which framework has better score. This is due to the fact that Δ(Max-Min) also includes the positive change of the R^2 score across incremental training size. Positive changes do not indicate over-fitting; therefore, Δ(Max Drop), which includes only negative changes, can justify how well each frameworks prevent over-fitting. According to Δ(Max Drop) columns, 76.5% of all the datasets show better Δ(Max Drop) value for Stacked auto_ml. In addition, Fig. 5.24 compares raw max drop values for both frameworks across all 50 datasets.The figure shows that Stacked auto_ml performs better than auto_ml in 37 out of 50 datasets which is 76%.

With the additional components to Stacked auto_ml, the runtime for the framework also slows down. Fig. 5.3 and Fig. 5.4 shows the R^2 score and training runtime of Stacked auto_ml and auto_ml for the first 17 datasets. The training runtime for Stacked auto_ml increase approximately a second on average for each dataset when compared to the auto_ml's training runtime. This is not an unreasonable delay considering the resilience against over-fitting.

One of the factors for the longer training time is the used of grid search, which is in the meta-learner training phase. Fig. 5.25 shows the runtime of using grid search and pseudo inverse to find the optimal hyper-parameters for the meta-learner. Since the meta-learner is a Linear Regression model, using pseudo inverse yields much faster runtime, as shown in the figure. The median improvement of using pseudo inverse is approximately 0.61 seconds faster than using the grid search. However, using grid search allows future expansions to more complicated meta-learners.

	Variance			Max - Min			Max Drop		
Datasets	**Stacked auto_ml**	**auto_ml**	**Δ**	**Stacked auto_ml**	**auto_ml**	**Δ**	**Stacked auto_ml**	**auto_ml**	**Δ**
data1	2.37E-03	1.90E-02	**-1.66E-02**	0.1699	0.4635	**-0.2935**	0.03471	0.40444	**-0.36973**
data2	2.76E-06	1.20E-06	1.56E-06	0.0054	0.0031	0.0023	0.00004	0.00093	**-0.00089**
data3	1.32E-03	1.59E-02	**-1.46E-02**	0.1131	0.2774	**-0.1643**	0.10817	0.27147	**-0.16329**
data4	2.85E-02	2.59E-04	2.82E-02	0.5263	0.0473	0.4790	0.34928	0.03141	0.31787
data5	3.78E-05	1.88E-04	**-1.50E-04**	0.0186	0.0489	**-0.0303**	0.00776	0.01138	**-0.00363**
data6	2.49E-04	3.45E-04	**-9.63E-05**	0.0535	0.0509	0.0026	0.00466	0.01175	**-0.00709**
data7	3.30E-05	2.62E-05	6.80E-06	0.0132	0.0163	**-0.0030**	0.00131	0.00722	**-0.00591**
data8	5.67E-05	4.45E-05	1.22E-05	0.0204	0.0214	**-0.0010**	0.00096	0.01106	**-0.01010**
data9	4.53E-08	4.18E-08	3.46E-09	0.0006	0.0007	0.0000	0.00011	0.00025	**-0.00014**
data10	3.13E-01	3.56E+00	**-3.25E+00**	1.4351	4.7825	**-3.3474**	1.06934	4.78247	**-3.71314**
data11	8.61E-04	6.59E-04	2.02E-04	0.0874	0.0724	0.0150	0.02704	0.01144	0.01559
data12	6.81E-04	1.12E-04	5.69E-04	0.0926	0.0268	0.0658	0.08942	0.00952	0.07990
data13	1.58E-04	2.36E-04	**-7.80E-05**	0.0321	0.0478	**-0.0157**	0.00223	0.01290	**-0.01067**
data14	4.15E-06	4.46E-05	**-4.05E-05**	0.0061	0.0228	**-0.0166**	0.00255	0.02004	**-0.01749**
data15	2.03E-03	3.71E-02	**-3.51E-02**	0.1439	0.4086	**-0.2646**	0.10049	0.38263	**-0.28214**
data16	1.86E-03	1.74E-04	1.69E-03	0.1302	0.0477	0.0826	0.04405	0.00286	0.04118
data17	4.66E-04	8.63E-04	**-3.97E-04**	0.0613	0.0992	**-0.0379**	0.01521	0.07515	**-0.05993**

Table 5.3: Results from comparing the R^2 score of Stacked auto_ml and auto_ml of each dataset.

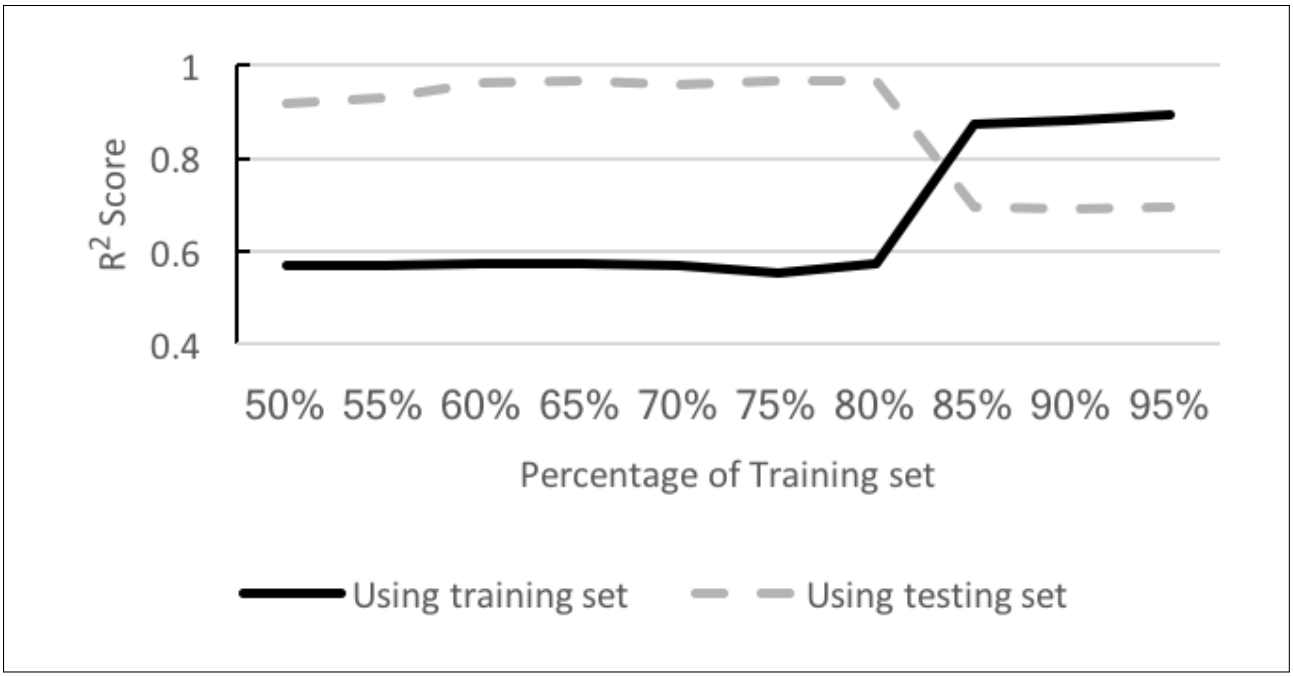

Figure 5.22: R^2 Scores of auto_ml predicting training set and testing set for data3 (Closer to 1 is better). Solid curve and dashed curve show how well the framework's predictions matched the training set and testing set respectively. Both cases use the same set of training space which results over-fitting.

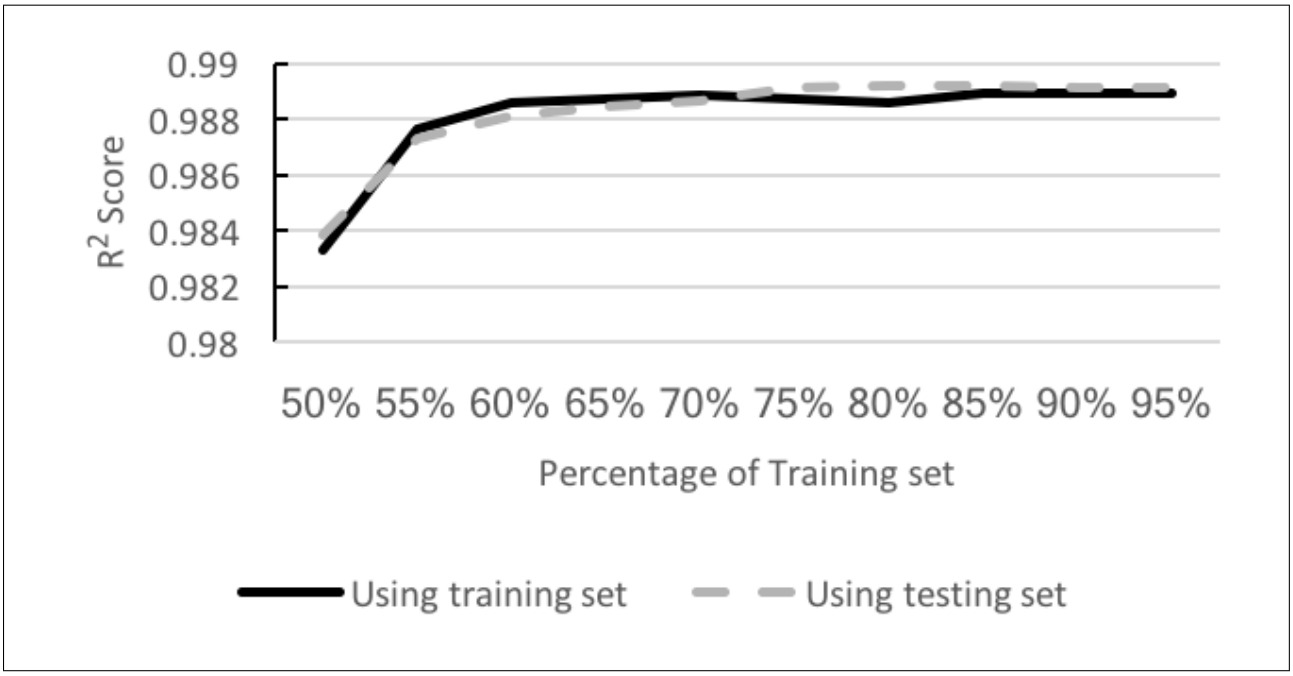

Figure 5.23: R^2 Scores of auto_ml predicting training set and testing set for data2 (Closer to 1 is better). Solid curve and dashed curve show how well the framework's predictions matched the training set and testing set respectively. Both cases use the same set of training space which does not result over-fitting.

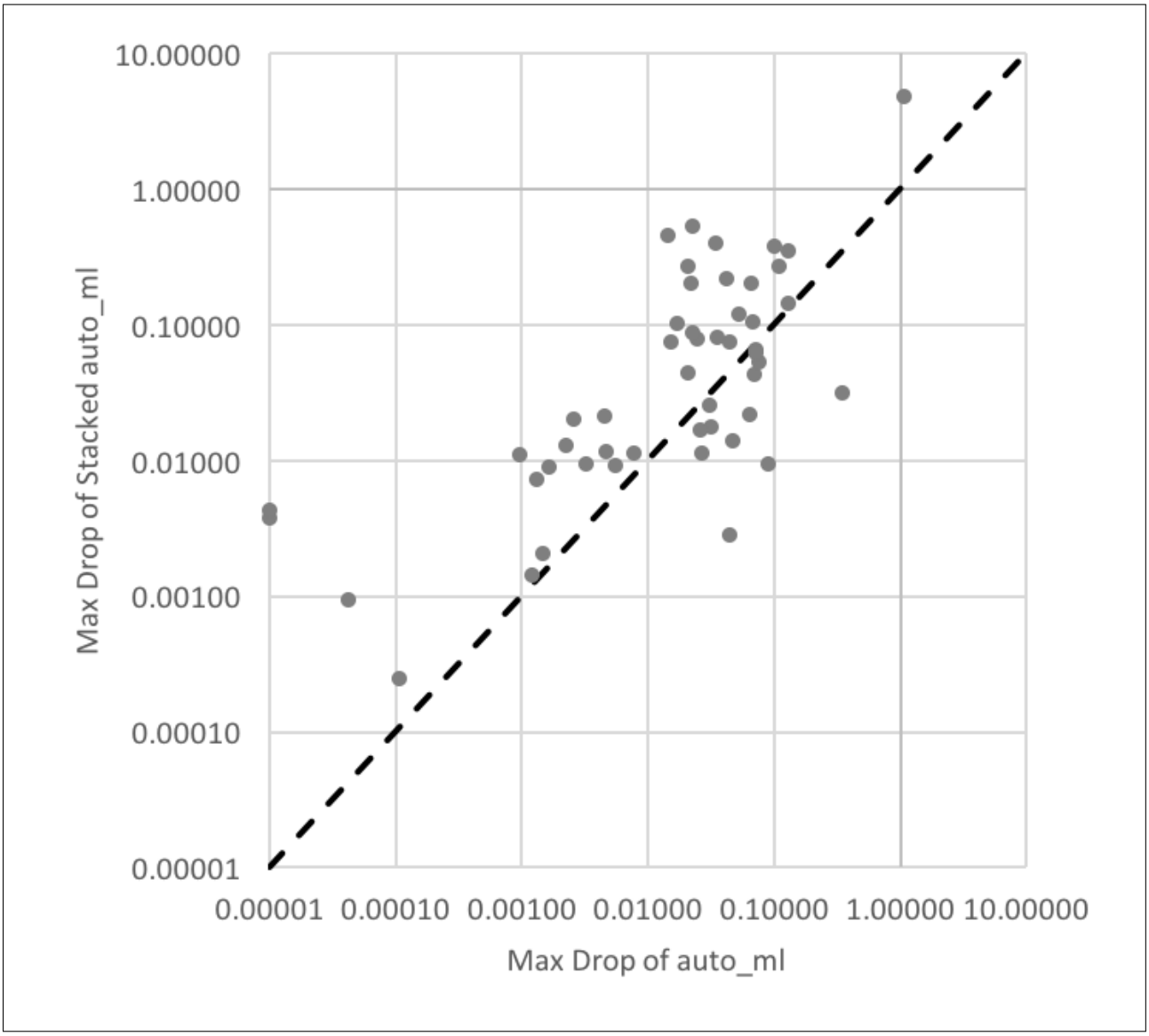

Figure 5.24: Max drop of Stacked auto_ml and auto_ml. Each dot represents a dataset. If a dot is above the dashed line, it means Stacked auto_ml has a lower max drop; hence more resilient against over-fitting. Both axes are in logarithmic scale. Stacked auto_ml performed better than auto_ml in 37 out of 50 datasets which is 76% of all testing data.

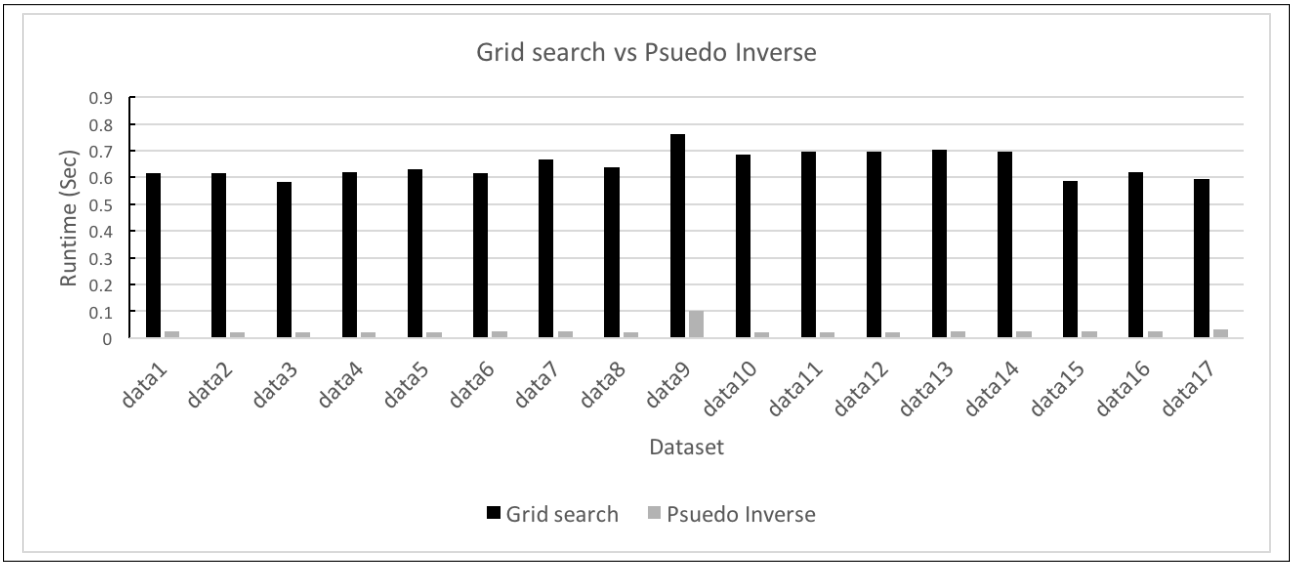

Figure 5.25: Runtime of grid search and pseudo inverse in Stacked auto_ml. This figure shows the runtime of both methods when used to find an optimal set of hyper-parameters for a Linear Regression model. The median improvement of using pseudo inverse is approximately 0.61 seconds faster than using the grid search.

5.4 Stacked auto_ml vs Regularization vs Dropout

This section tests the performance of using Stacked auto_ml and MLP model with L2 regularization and dropout for the first 17 datasets. This test goal is to understand how these techniques prevent over-fitting in small datasets. That's why, while the accuracy is important for any machine learning model, this test focuses mainly on the consistency of each technique. Similarly to Section 5.3, the same tests are used to compare the performance of these framework. For this test, the α value for L2 regularization is set to 1, and the dropout possibility is to 20%. The α value is expressed in (1.3), (1.4), and (1.5). Figs. 5.26 to 5.42 shows the R^2 of the first 17 datasets. Stacked auto_ml performed better than the two other techniques in both accuracy and consistency in 14 out of 17 datasets. Only Fig. 5.40 shows MLP with dropout outperformed Stacked auto_ml. The other two Figs. 5.27 and Fig. 5.37 show the performance of Stacked auto_ml and MLP with L2 Regularization are very close; therefore, a conclusion cannot be drawn on which has better performance. The results have shown that, for small datasets, Stacked auto_ml is more resilient to over-fitting than MLP with L2 regularization and dropout.

This test also looks at the runtime of each technique. For each dataset, the runtime is calculated by taking the average of all training runtime across multiple training space percentage. Fig. 5.43 shows the runtime of Stacked auto_ml, MLP with L2 regularization, and MLP with dropout. In all datasets, MLP with L2 regularization is the fastest to finish its training process. On average, MLP with L2 regularization takes under a second to finish its training. Stacked auto_ml's training time is slower, but the training time stays consistently below 10 seconds, on average. It also runs faster than MLP with dropout in 6 out of 17 datasets. Considering that Stacked auto_ml also performs grid search, and other data reprocesses, its runtime is negligible.

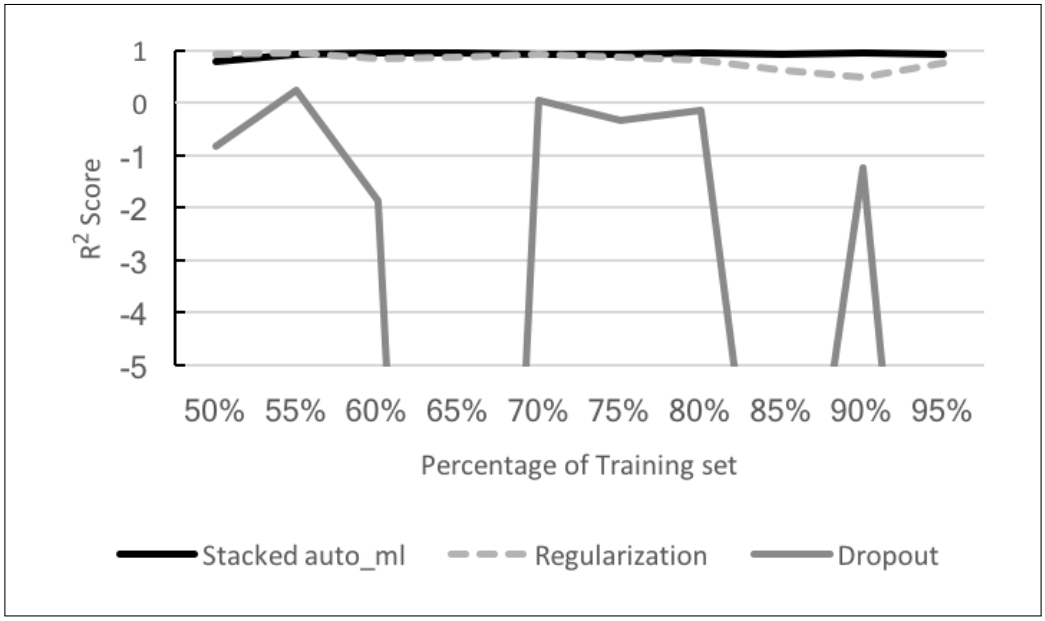

Figure 5.26: R^2 Scores of Stacked auto_ml, MLP with L2 regularization, and MLP with dropout for data1 (Closer to 1 is better). Some data points might be omitted because they are too large.

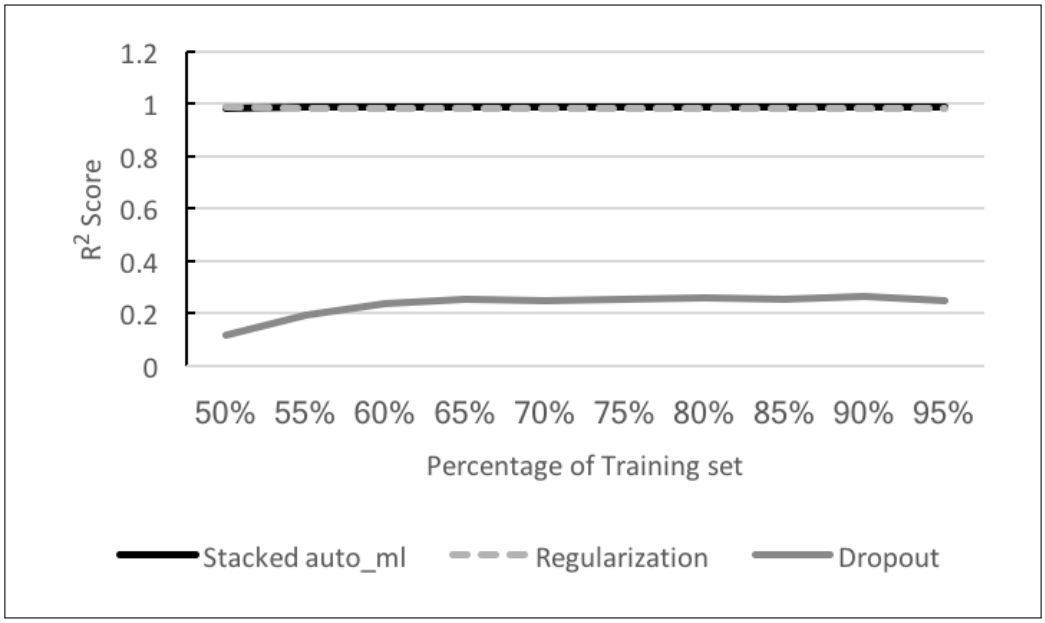

Figure 5.27: R^2 Scores of Stacked auto_ml, MLP with L2 regularization, and MLP with dropout for data2 (Closer to 1 is better). Some data points might be omitted because they are too large.

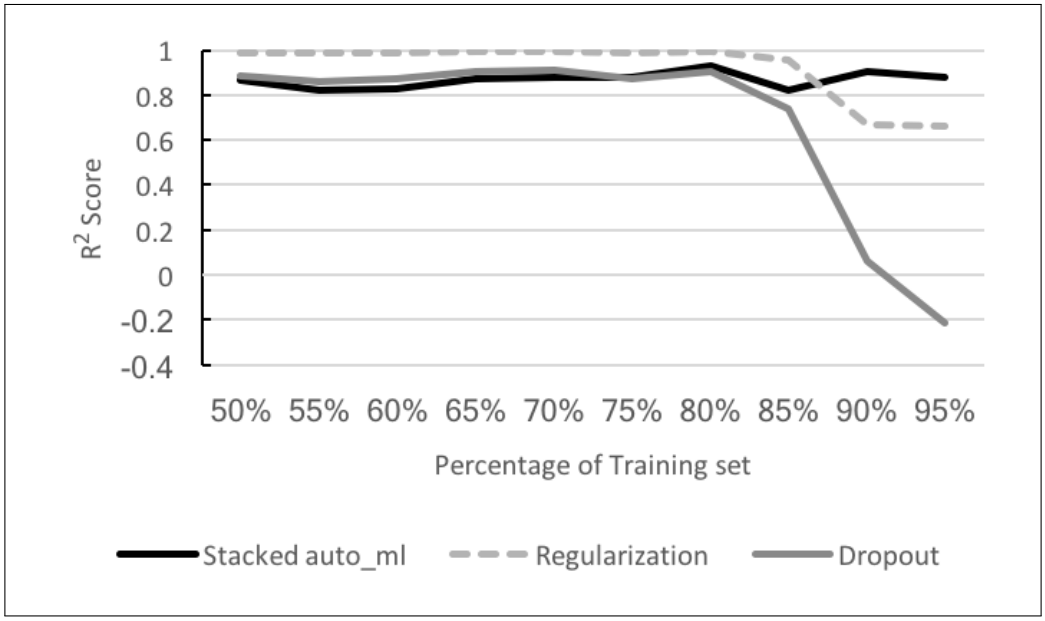

Figure 5.28: R^2 Scores of Stacked auto_ml, MLP with L2 regularization, and MLP with dropout for data3 (Closer to 1 is better). Some data points might be omitted because they are too large.

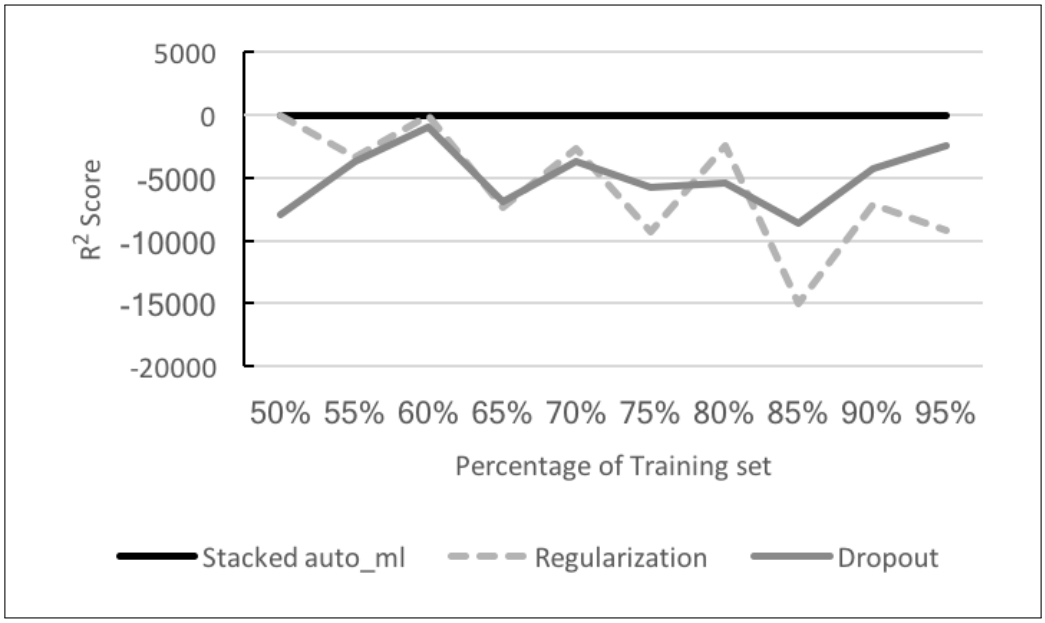

Figure 5.29: R^2 Scores of Stacked auto_ml, MLP with L2 regularization, and MLP with dropout for data4 (Closer to 1 is better). Some data points might be omitted because they are too large.

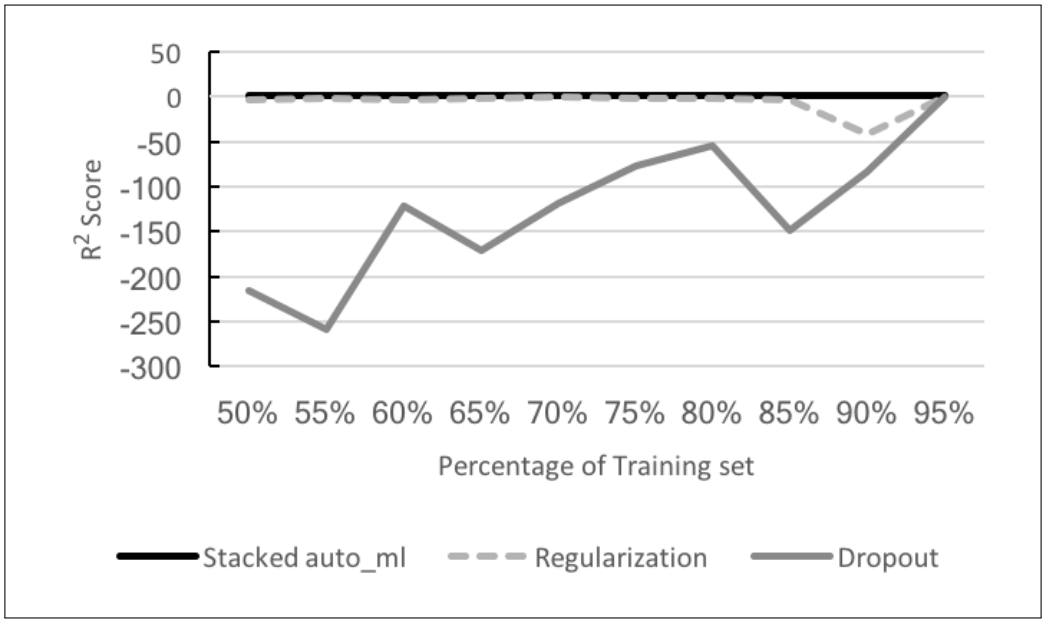

Figure 5.30: R^2 Scores of Stacked auto_ml, MLP with L2 regularization, and MLP with dropout for data5 (Closer to 1 is better). Some data points might be omitted because they are too large.

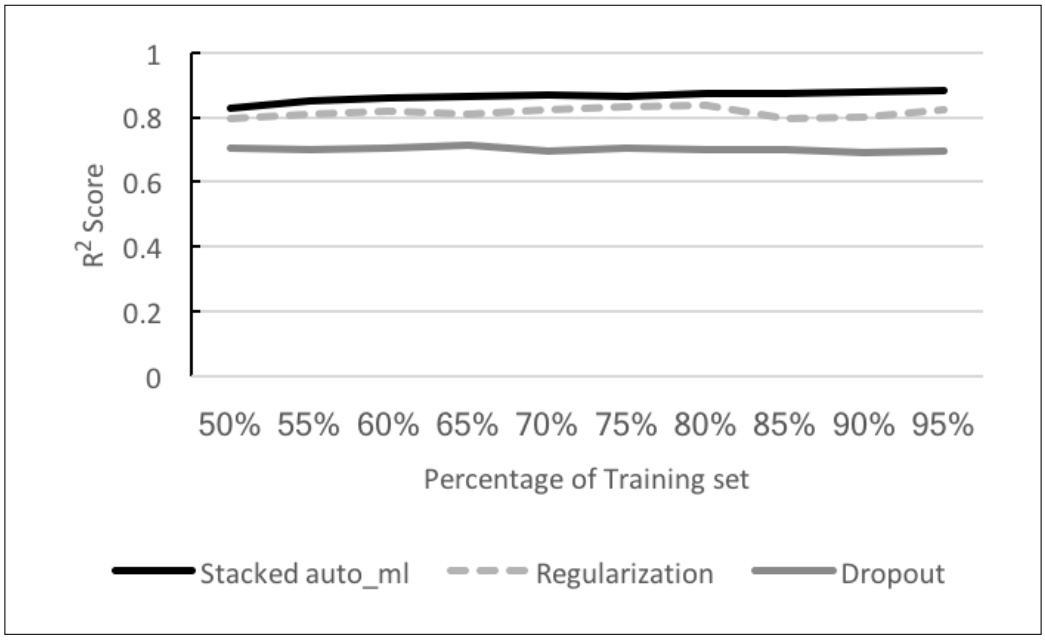

Figure 5.31: R^2 Scores of Stacked auto_ml, MLP with L2 regularization, and MLP with dropout for data6 (Closer to 1 is better). Some data points might be omitted because they are too large.

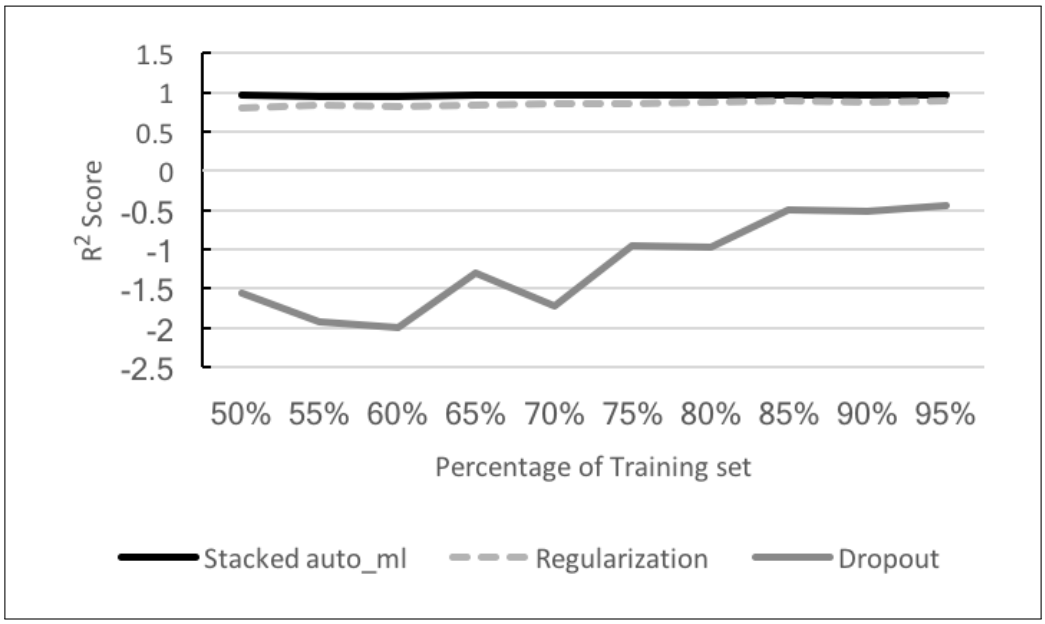

Figure 5.32: R^2 Scores of Stacked auto_ml, MLP with L2 regularization, and MLP with dropout for data7 (Closer to 1 is better). Some data points might be omitted because they are too large.

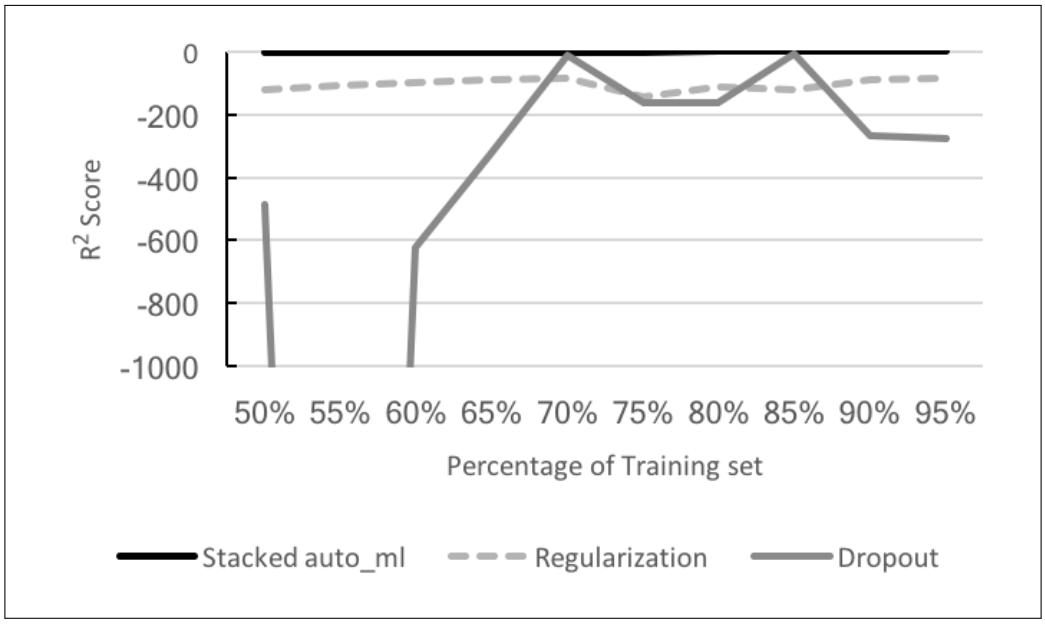

Figure 5.33: R^2 Scores of Stacked auto_ml, MLP with L2 regularization, and MLP with dropout for data8 (Closer to 1 is better). Some data points might be omitted because they are too large.

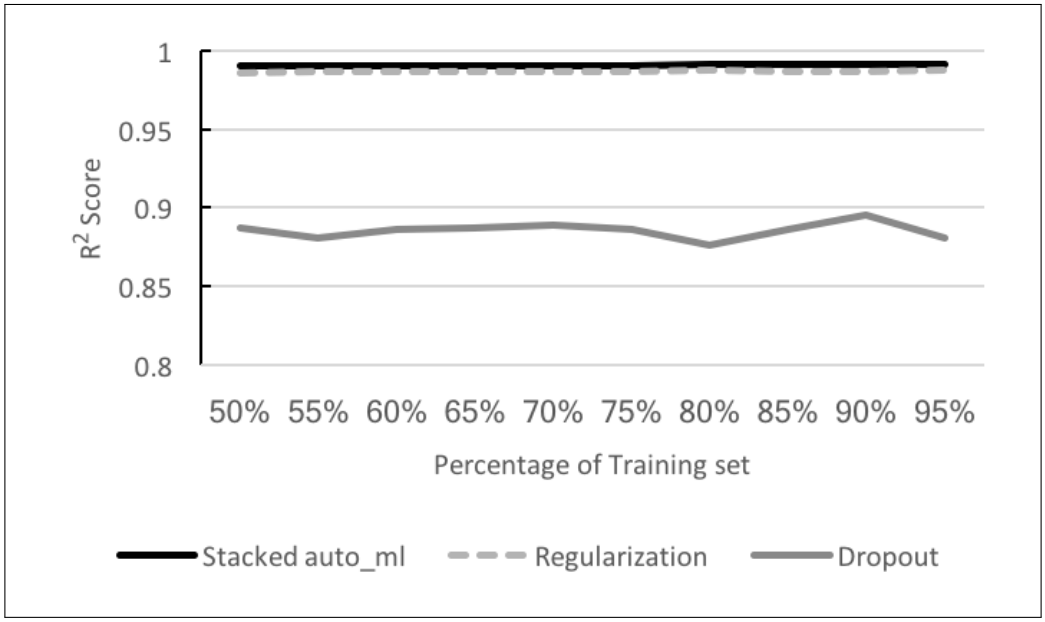

Figure 5.34: R^2 Scores of Stacked auto_ml, MLP with L2 regularization, and MLP with dropout for data9 (Closer to 1 is better). Some data points might be omitted because they are too large.

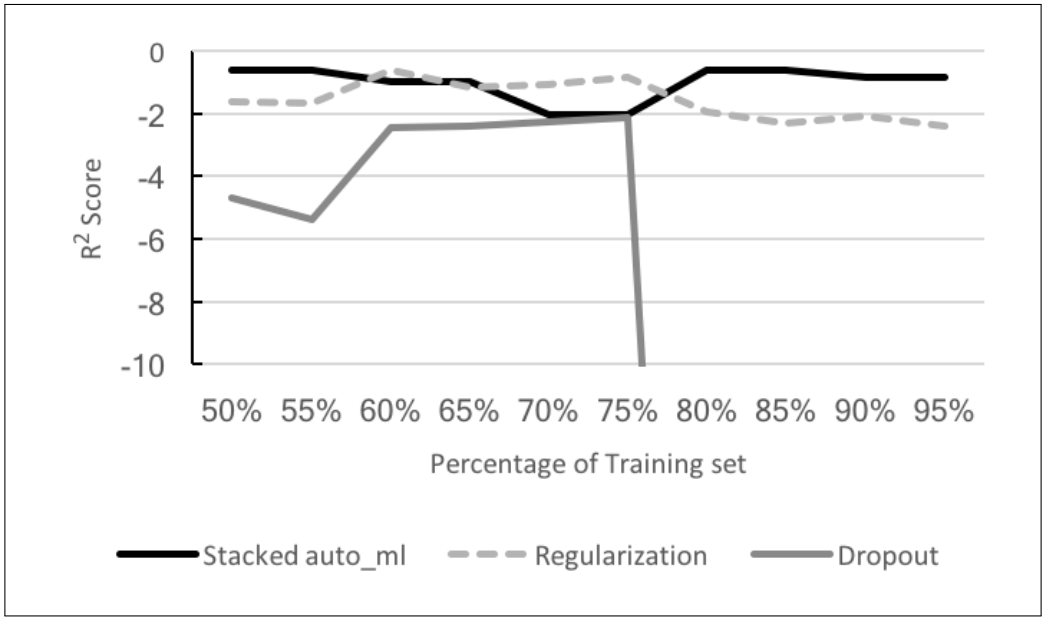

Figure 5.35: R^2 Scores of Stacked auto_ml, MLP with L2 regularization, and MLP with dropout for data10 (Closer to 1 is better). Some data points might be omitted because they are too large.

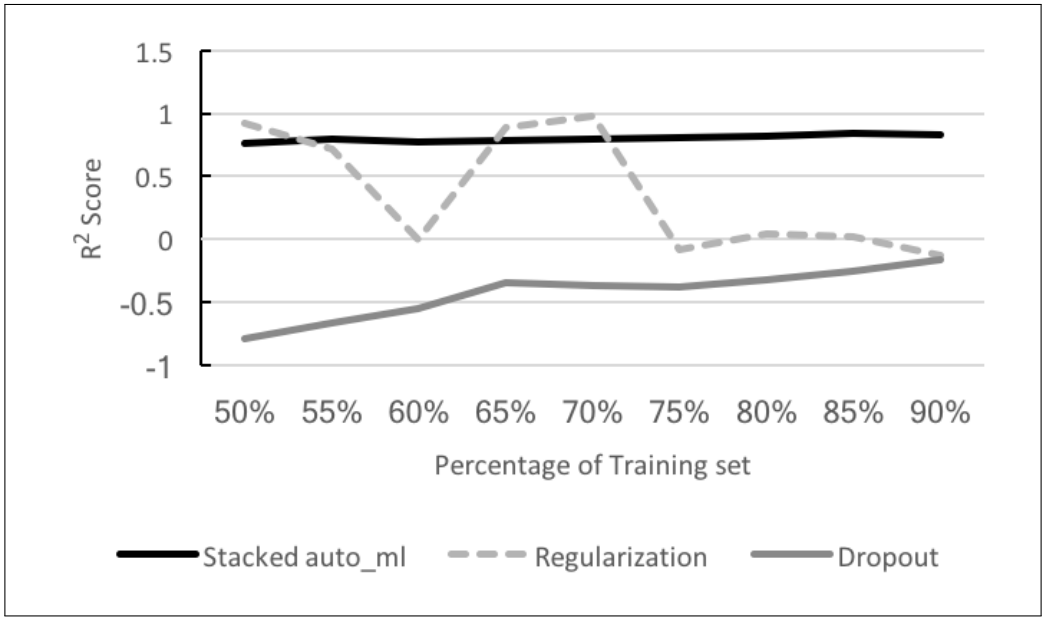

Figure 5.36: R^2 Scores of Stacked auto_ml, MLP with L2 regularization, and MLP with dropout for data11 (Closer to 1 is better). Some data points might be omitted because they are too large.

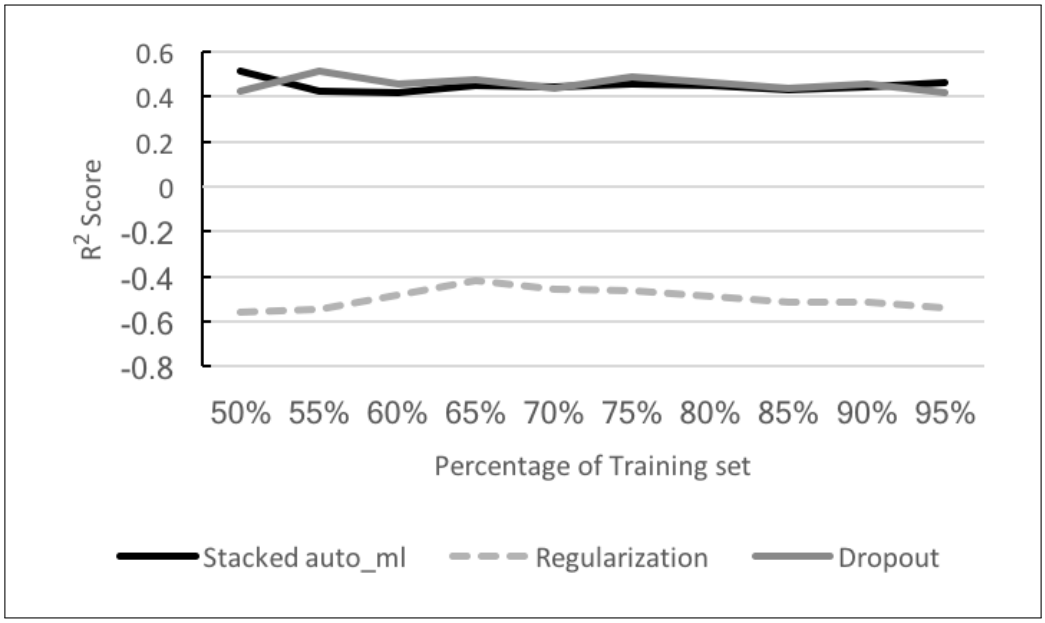

Figure 5.37: R^2 Scores of Stacked auto_ml, MLP with L2 regularization, and MLP with dropout for data12 (Closer to 1 is better). Some data points might be omitted because they are too large.

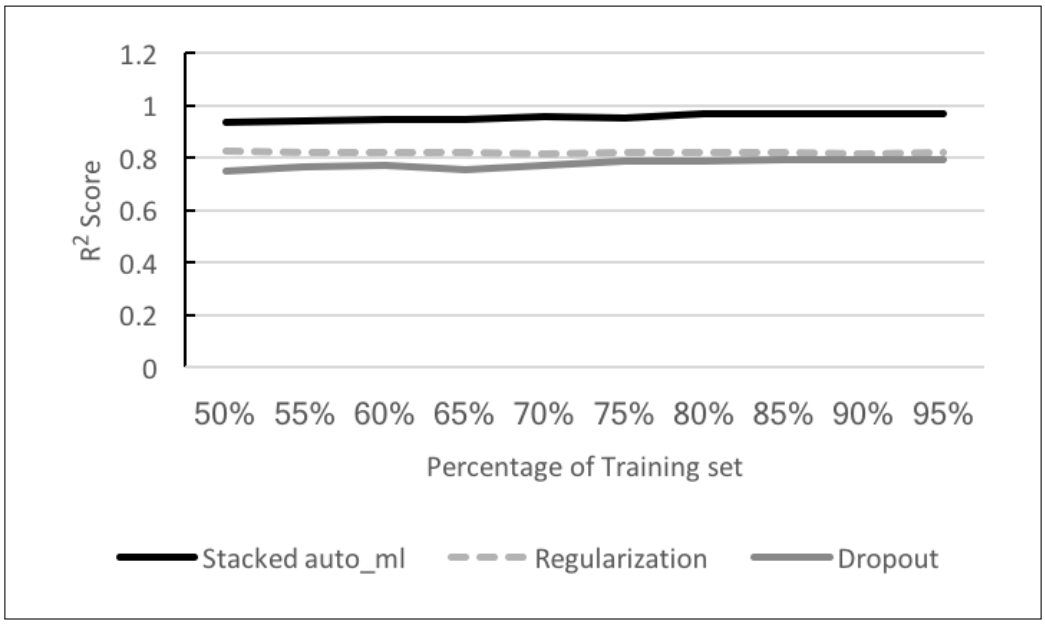

Figure 5.38: R^2 Scores of Stacked auto_ml, MLP with L2 regularization, and MLP with dropout for data13 (Closer to 1 is better). Some data points might be omitted because they are too large.

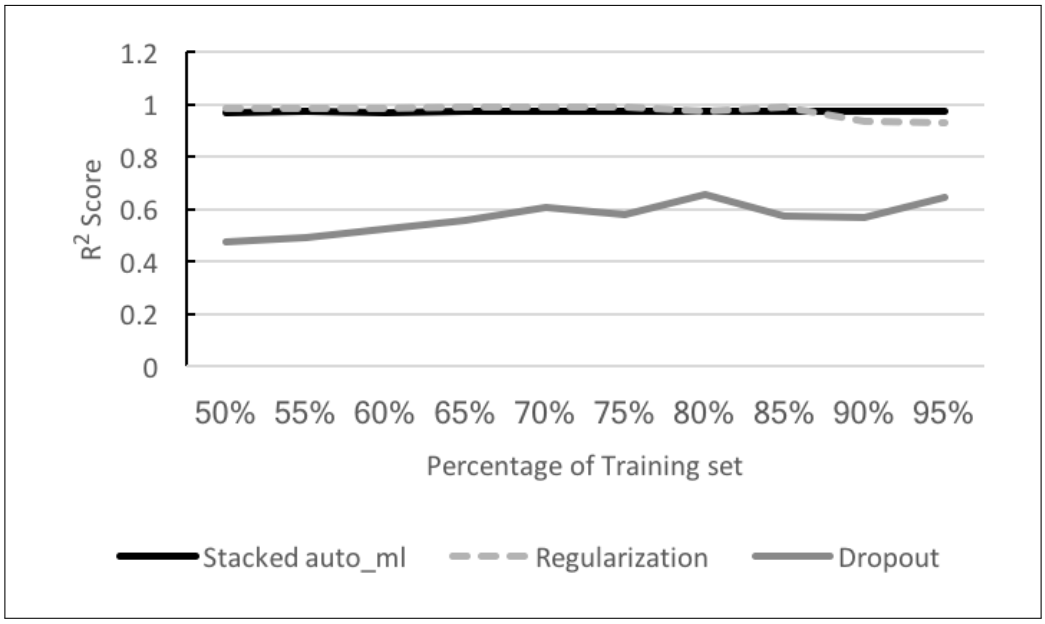

Figure 5.39: R^2 Scores of Stacked auto_ml, MLP with L2 regularization, and MLP with dropout for data14 (Closer to 1 is better). Some data points might be omitted because they are too large.

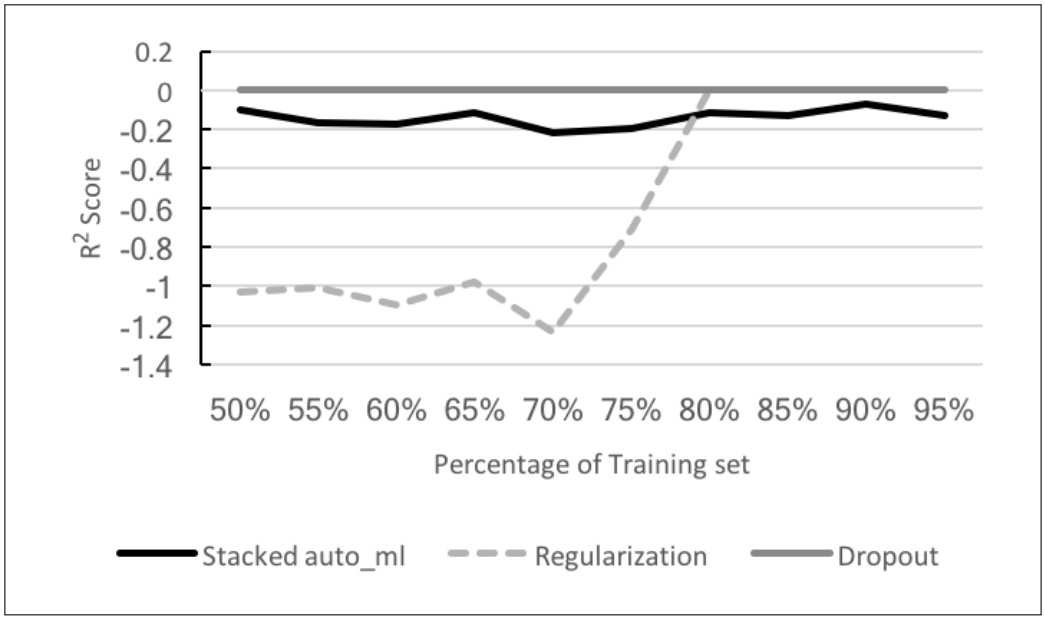

Figure 5.40: R^2 Scores of Stacked auto_ml, MLP with L2 regularization, and MLP with dropout for data15 (Closer to 1 is better). Some data points might be omitted because they are too large.

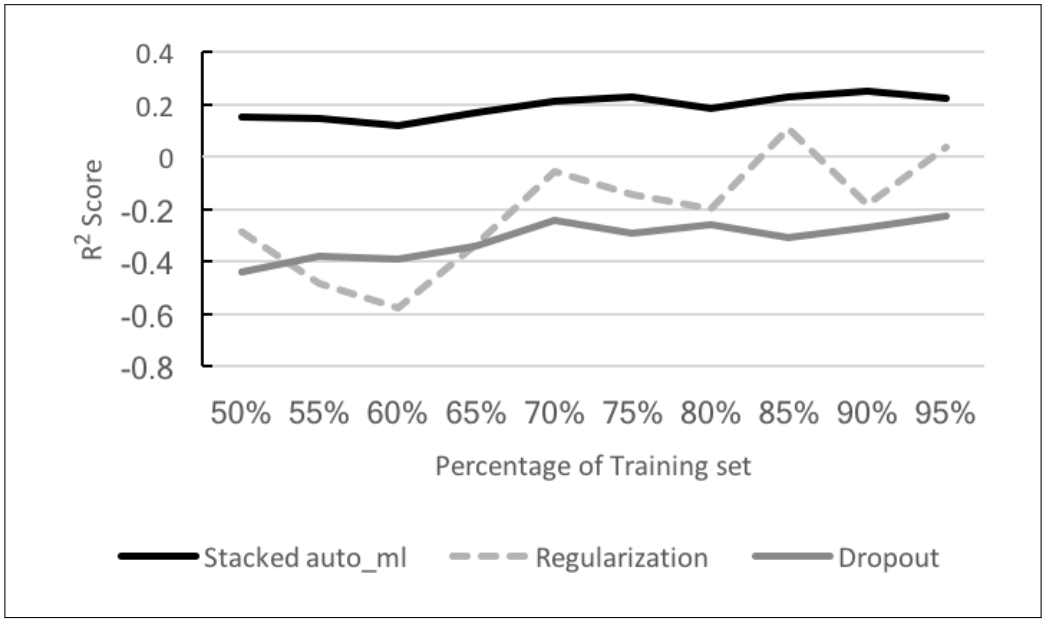

Figure 5.41: R^2 Scores of Stacked auto_ml, MLP with L2 regularization, and MLP with dropout for data16 (Closer to 1 is better). Some data points might be omitted because they are too large.

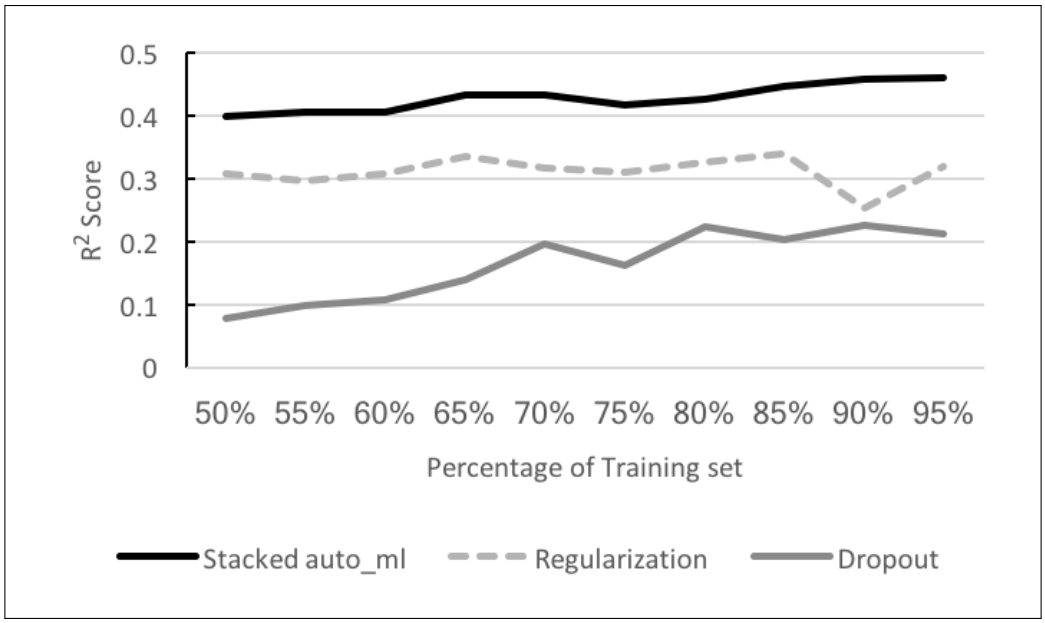

Figure 5.42: R^2 Scores of Stacked auto_ml, MLP with L2 regularization, and MLP with dropout for data17 (Closer to 1 is better). Some data points might be omitted because they are too large.

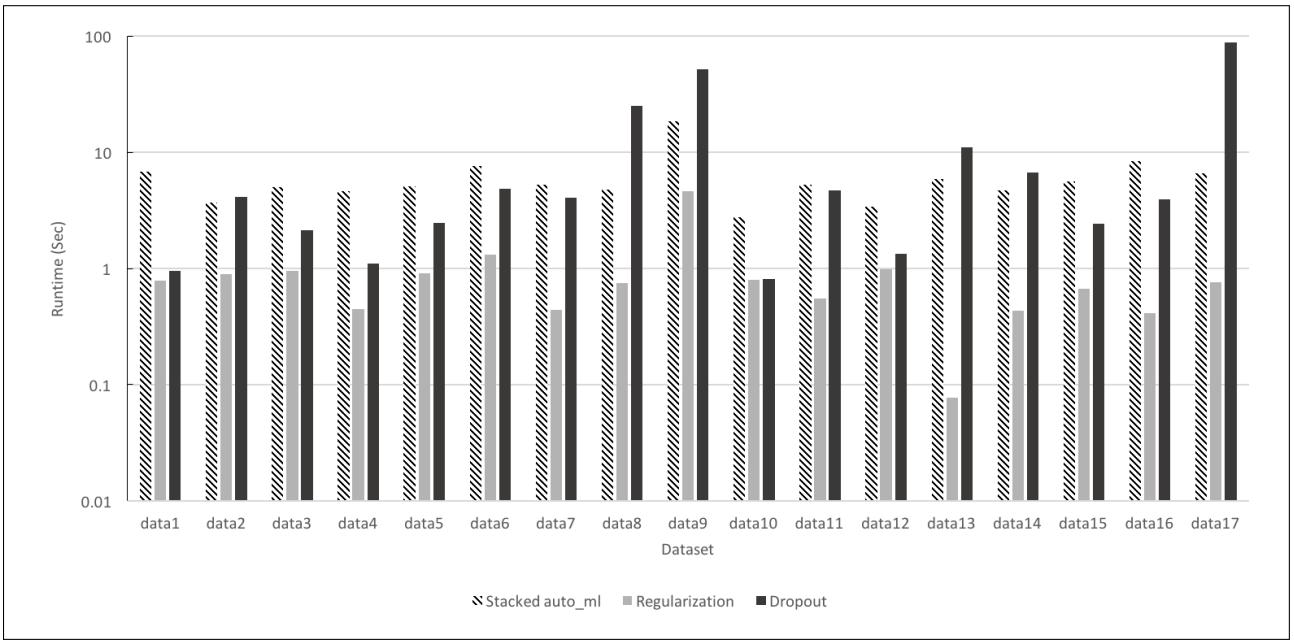

Figure 5.43: Training runtime of Stacked auto_ml, MLP with L2 regularization, and MLP with dropout.

5.5 Stacked auto_ml vs Individual Base Model

This test verifies the performance of Stacked auto_ml to individual base model used in the framework's implementation. Fig. 5.44 shows the R^2 scores of the framework compared to four base models: Linear Regression model, Extra Trees Regression model, Random Forest Regression model, and Gradient Boosting Regression model. As shown in the figure, the framework performs very close to the the best model in each dataset and the scores are never the worst performing scores. We can conclude that the framework guarantees good consistent performance across different datasets.

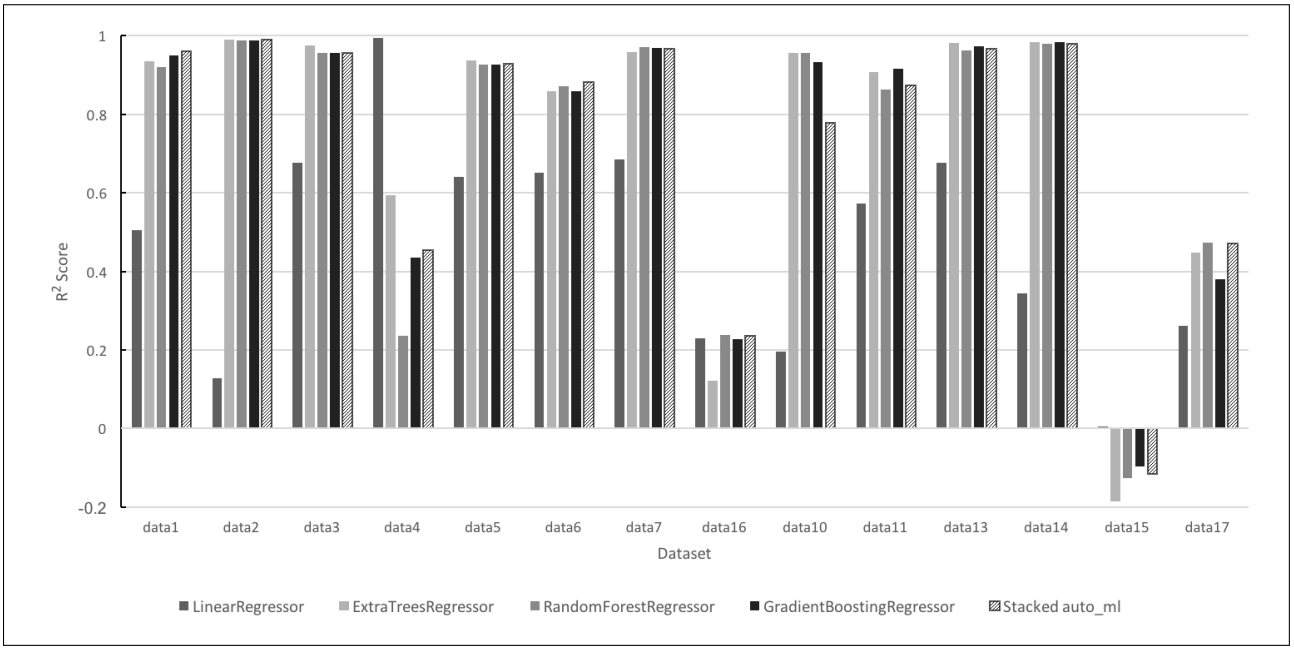

Figure 5.44: R^2 Scores of Stacked auto_ml and individual base models (Closer to 1 is better).

5.6 Stacked auto_ml using Whole Data vs Splitting Ratios

This test verifies the performance of Stacked auto_ml using different splitting ratios. As stated in the Stacking Implementation Section, there are other ways to split data to train the base models and the meta-learner. For this test, training data is split three different ratios to train base models and train meta-learner: 7:3, 6:4, and 5:5. These different ratios are used to compared using the whole training data for both training base models and meta-learner. Fig. 5.45 shows that using whole data performs much better in the majority of datasets. This proves that using whole data to train both base models and meta-learner is appropriated for small datasets.

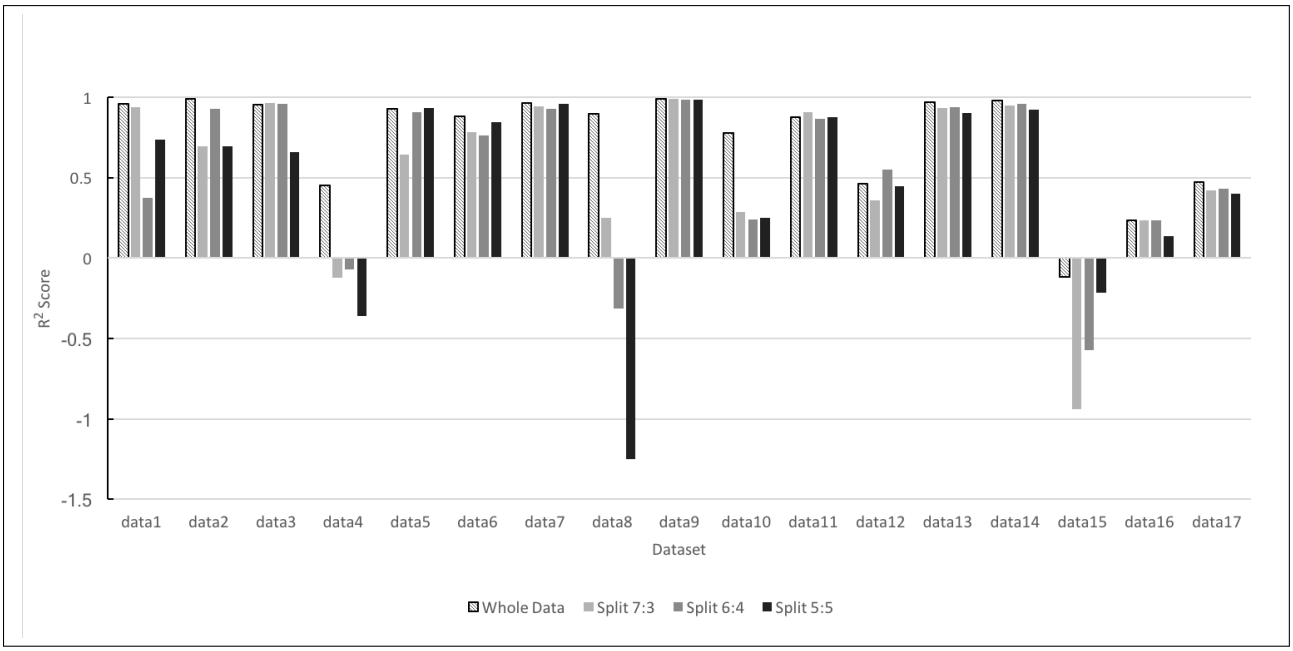

Figure 5.45: R^2 Scores of different splitting ratios for Stacked auto_ml.

Chapter 6

Summary

6.1 Future Works

This thesis introduces a simple version of Stacking; therefore, there are not much of an improvement in performance over the original framework. With a more complex Stacking structure, such as more ensemble layers, an improvement could be made. It is also possible to expand the list of models, instead of using a fixed set. Moreover, an algorithm can be implemented to choose which models to be trained and use. This can reduce the computational time. The thesis does not cover and analyze big dataset, which is also an important part of Machine Learning. The difference of small and big datasets is that small datasets do not have enough data to robustly train a model. On the other hand, big datasets provides enough data to build robust models that consistently get excellent performance. Therefore, the framework must improve the performance instead of the performance consistency.

6.2 Conclusion

Machine learning is a fast growing field that is very beneficial to various industries, such as researching, online retailing, advertisement, medical, politics, and social media. Because of that, there are more inexperienced developers, who are not deeply familiar with machine learning, want to take advantages of this concept. The thesis introduces auto_ml and a

Stacking implementation to the framework. The auto_ml framework has shown that it is a reliable framework to predict different kind of data, especially small datasets. It is shown that auto_ml can perform on par with auto-sklearn and only uses a fraction of the time. With Stacking, auto_ml can further ensure the reliability against over-fitting; hence, it results in good and consistent performance. Stacked auto_ml performs more consistently than auto_ml in 76% of the tested datasets. In addition, any developers can fully enjoy the advantage of Machine Learning models and tools because the framework requires little to no Machine Learning background. The ultimate goal is to spread the popularity of Machine Learning to more people for different application usages. The framework can be beneficial to different areas beside research and computing, such as medical, politics, and social media.

Bibliography

[1] MP Austin, L Belbin, JA Meyers, MD Doherty, and M Luoto. Evaluation of statistical models used for predicting plant species distributions: role of artificial data and theory. *ecological modelling*, 199(2):197–216, 2006.

[2] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(Feb):281–305, 2012.

[3] James S Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In *Advances in neural information processing systems*, pages 2546–2554, 2011.

[4] Leo Breiman. Bias, variance, and arcing classifiers. 1996.

[5] Rich Caruana, Alexandru Niculescu-Mizil, Geoff Crew, and Alex Ksikes. Ensemble selection from libraries of models. In *Proceedings of the twenty-first international conference on Machine learning*, page 18. ACM, 2004.

[6] Yijun Chen and Man Leung Wong. An ant colony optimization approach for stacking ensemble. In *Nature and Biologically Inspired Computing (NaBIC), 2010 Second World Congress on*, pages 146–151. IEEE, 2010.

[7] Robert Daber, Shrey Sukhadia, and Jennifer JD Morrissette. Understanding the limitations of next generation sequencing informatics, an approach to clinical pipeline validation using artificial data sets. *Cancer genetics*, 206(12):441–448, 2013.

[8] Saso Džeroski and Bernard Ženko. Is combining classifiers with stacking better than selecting the best one? *Machine learning*, 54(3):255–273, 2004.

[9] Pierre Geurts, Damien Ernst, and Louis Wehenkel. Extremely randomized trees. *Machine learning*, 63(1):3–42, 2006.

[10] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H Witten. The weka data mining software: an update. *ACM SIGKDD explorations newsletter*, 11(1):10–18, 2009.

[11] Clifford M Hurvich and Chih-Ling Tsai. Regression and time series model selection in small samples. *Biometrika*, 76(2):297–307, 1989.

[12] Yann LeCun, LD Jackel, Léon Bottou, Corinna Cortes, John S Denker, Harris Drucker, Isabelle Guyon, UA Muller, Eduard Sackinger, Patrice Simard, et al. Learning algorithms for classification: A comparison on handwritten digit recognition. *Neural networks: the statistical mechanics perspective*, 261:276, 1995.

[13] Maxwell W Libbrecht and William Stafford Noble. Machine learning applications in genetics and genomics. *Nature Reviews Genetics*, 16(6):321, 2015.

[14] M. Lichman. UCI machine learning repository, 2013. URL http://archive.ics.uci.edu/ml.

[15] Jianchang Mao. A case study on bagging, boosting and basic ensembles of neural networks for ocr. In *Neural Networks Proceedings, 1998. IEEE World Congress on Computational Intelligence. The 1998 IEEE International Joint Conference on*, volume 3, pages 1828–1833. IEEE, 1998.

[16] Nico JD Nagelkerke et al. A note on a general definition of the coefficient of determination. *Biometrika*, 78(3):691–692, 1991.

[17] Sajid Nagi and Dhruba Kr Bhattacharyya. Classification of microarray cancer data

using ensemble approach. *Network Modeling Analysis in Health Informatics and Bioinformatics*, 2(3):159–173, 2013.

[18] Andrew Y Ng. Feature selection, l 1 vs. l 2 regularization, and rotational invariance. In *Proceedings of the twenty-first international conference on Machine learning*, page 78. ACM, 2004.

[19] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[20] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of machine learning research*, 12(Oct):2825–2830, 2011.

[21] S Rasoul Safavian and David Landgrebe. A survey of decision tree classifier methodology. *IEEE transactions on systems, man, and cybernetics*, 21(3):660–674, 1991.

[22] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*, pages 2951–2959, 2012.

[23] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[24] Daniel Svozil, Vladimir Kvasnicka, and Jiri Pospichal. Introduction to multi-layer feed-

forward neural networks. *Chemometrics and intelligent laboratory systems*, 39(1):43–62, 1997.

[25] Larry Winner. Larry winner's data, 2011. URL http://users.stat.ufl.edu/~winner/data/.

[26] David H Wolpert. Stacked generalization. *Neural networks*, 5(2):241–259, 1992.