DELAWARE STATE UNIVERSITY

MASTERS THESIS

A comparison of meta-learning strategies

Author:
John Liddell

Supervisor:
Dr. Tomasz Smolinski

A thesis submitted in fulfillment of the requirements for the degree of Masters in Science

in the

Department of computer science

October 29, 2019

Acknowledgements

I would like to acknowledge those who enabled the completion of this thesis, Dr. Smolinski, Dr. Holness, the staff of the computer science department and my committee group. Without you all I could not have completed this thesis. Thank you all for the guidance and support.

Contents

List of Tables

DELAWARE STATE UNIVERSITY

Abstract

A comparison of meta-learning strategies

by John Liddell

A comparison of several meta-learning strategies in order to ascertain the truth of the “no free lunch theorem,” as it applies to metalearning strategies, was performed. The data needed in order to validate the performance of each meta-learning strategy were gathered from the UCI Irvine dataset repository, parsed, then stored in a database. Collections of datasets were then created within this database, and the meta-learning strategies were compared in terms of their accuracies with relation to these collections. t score analysis was then performed in order to determine whether the relative accuracies of the meta-learning algorithms were within expectation.

Chapter 1

Introduction

Determining what algorithm to use when analyzing a dataset is a problem as old as machine learning itself. In “No free lunch theorems for optimization” Wolpert and Macready demonstrate that the relative performance of any two given machine learning algorithms will be uniform across all datasets, that is to say a machine learning algorithms performance is contingent on the problem space in which the algorithm is operating. As such, the decision of algorithm is not arbitrary and some strategy must be employed in order to decide on an algorithm. In some cases, the individuals wishing to perform an analysis have access to an expert, possibly themselves, that can simply tell them which algorithm is best in the given situation. In other situations, the individuals wishing to perform analysis may not have the budget necessary to acquire access to such an expert, in which case the usage of a meta-learner becomes appropriate. With a meta-learner, one feeds the meta-learner a dataset, and it returns to the user what it thinks is the most appropriate machine with which to perform analysis. To get to the point wherein a decision can be made on new datasets, the meta-learner itself must first be trained, and this training requires some sort of learning strategy. This fact suggests that the decision of what meta-learning strategy to use for some given body of datasets should be susceptible to the previously mentioned no free lunch theorem, that is to say that some meta-learning strategies will work better on some given set of databases than others. The confirmation or denial of this theorem in this context is the goal of this thesis. Including the current one, this thesis is comprised of five chapters. In Chapter 2, a review of the base machine and meta learning strategies used within the experiment is done. Chapter 3 describes the structure of the experiments code at a high level. Chapter 4 analyzes the results table in order to determine whether or not one meta-learning strategy strictly dominates. Chapter 5 presents a summary of the document and addresses a possible weakness in the results.

Chapter 2

Review of the literature

2.1 No Free Lunch Theorem

Introduced in Wolpert and Macready's "No Free Lunch Theorems for Optimization" 1997 paper, the No Free Lunch theorem states that the performance of all algorithms, when averaged out across all datasets, should be the same; that is to say there is no one algorithm that is universally the best. The root cause of this phenomenon is in that differing algorithms make different assumptions about the distributions from which the data the algorithms work with arises. A learning algorithm with an implicit assumption of a random distribution will have a far lower test case classification accuracy than an algorithm that assumes a Gaussian distribution if the distribution from which the set of observed samples derives is truly normal and vice versa, if the distribution is truly random, the Gaussian classifier's accuracy will suffer relative to the classifier with a random assumption.

2.1.1 Brute Force Metabase

The most basic meta-machine learning algorithm. The accuracies of the meta-learners producible machines for some metabase are gathered. To classify a new dataset d_n, a clustering algorithm (K-Means in the case of this experiment) is used to find d_m, the dataset within the metabase with which the new dataset d_n is most similar. The algorithm which had the greatest classification accuracy for the metabase dataset d_m will then be returned by the meta-learner.

2.1.2 Active Meta Learning

The second of the meta-learning strategies implemented within the study, Active Meta Learning is a meta learning technique "that reduces the cost of generating Meta examples by selecting relevant Meta examples" [1]. What this entails is a decision on what datasets to allow into a meta-learner's metabase. Rather than analyze every candidate meta base dataset, an active meta-learner will analyze the next dataset with the highest uncertainty. The relative uncertainty between two datasets is defined to be:

$$\delta(V_x, d_i, V_x, d_j) = \frac{|V_x, d_i - V_x, d_j|}{Max_{k \neq i}(V_x, d_k) - Min_{k \neq i}(V_x, d_k)}$$

where V_x, d_k is the value of some metaparameter V_x for dataset d_k, $Max_{k \neq i}(V_x, d_k)$ is the maximum value of V_x, d_k when dataset i is removed and $Min_{k \neq i}(V_x, d_k)$ is its corresponding minimum. Determining which dataset has the overall highest uncertainty can be done via the following procedure. First, sum the relative uncertainties for each dataset and meta-parameter combination. Then, rank the uncertainty scores of the datasets within each meta-parameter. After obtaining the uncertainty ranks within each parameter for each dataset, sum the parameter ranks in order to obtain an overall uncertainty rank for each dataset. Finally, select the parameter with the highest rank for inclusion in the metabase. The equation representing the overall uncertainty score in a specific metaparameter V_x for dataset d_i is

$$\delta(V_x, d_i) = \frac{\sum_{j \neq i} |V_x, d_i - V_x, d_j|}{Max_{k \neq i}(V_x, d_k) - Min_{k \neq i}(V_x, d_k)}$$

2.1.3 Predicting Relative Performance of Classifiers from Samples

The third of the meta-learning strategies implemented within this study is one in which a representative subsection of the metabase is trained with each algorithm entirely, after which point the rest of the metabase undergoes curve sampling analysis; the accuracies of the base algorithms are predicted from run curve similarity rather than directly ran [2]. As with the other two meta learning strategies, the label for new datsets is then determined via clustering with the datasets contained within the metabase.

2.2 Summary of Producible Machines

The strategies mentioned in the previous section all consume a vector representation of some dataset, and then make a guess as to what algorithm would best be able to classify its data. The machines that these strategies can choose from are the K-means clustering algorithm, a neural network, a naive Bayes classifier, the support vector machine, and regression; with the results coming from the regression machine being cast into classificatory bins from the real valued result that it would produce. An in depth description of each of these different learning algorithms will comprise the rest of this chapter.

2.3 Linear Regression

Linear regression is one of the most common and oldest machine learning techniques. It asserts that the response is a linear function of the inputs [7]. This relation takes the following form:

$$y(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + \epsilon = \sum_{j=1}^{D} w_j x_j + \epsilon$$

where $w^T x$ represents the inner or scalar product between the input vector x and the model's weight vector w^T, and ϵ is the residual error between our linear predictions and the true response.

To fit a linear regression model, the least squares approach is usually used. Given some "overdetermined" linear system (that is to say a system in which there are more data points than parameters), one can write an expression for the sum of squares of the system

$$S(\beta) = (y_1 - \beta x_1)^2 + (y_2 - \beta x_2)^2 + ...(y_n - \beta x_n)^n$$

and then take the partial derivative of this sum of squares deviations with respect to each of the components of β, set them to zero, then solve the resulting equations to directly determine the values of the parameters that minimize the sum of the squared errors of the system. With linear regression in two dimensions (one dimension in the Independent variable and one dimension in the Dependent variable, we see a system with two parameters $\beta_0 = yintercept$ and $\beta_1 = slope$. If we had, for example, 3 data points (2,1),(3,7), and (4,5) we would have the equations $\beta_0 + 2 * \beta_1 = 1$, $\beta_0 + 3 * \beta_1 = 7$, and $\beta_0 + 4 * \beta_1 = 5$. The sum of squared errors would then be $S(\beta_0, \beta_1) = [1 - (\beta_0 + 2 * \beta_1)]^2 + [7 - (\beta_0 + 3 * \beta_1)]^2 + [5 - (\beta_0 + 4 * \beta_1)]^2$,which we could then differentiate with respect to β_0 and β_1 then directly solve the resulting set of linear equations directly for the minimum of the summed squares.

2.4 Naive Bayes

The Naive Bayes classifier algorithm fits a set of data to Bayes' Theorem with a strong assumption of feature independence. Given a set of discrete-valued features $x \in 1, ..., K^D$, we can calculate the class conditional density for each feature, then, with our assumption of independence, generate a guess at what the class should be for a new input by multiplying the conditional likelihood values for each of the new inputs features times the prior on the desired to be known class, that is to say $p(y|\mathbf{x}) \propto p(y) \sum_j^D p(x_j|y)$. The calculation of the posterior probability for a new example can be done manually, or can be derived from distributions that are inferred from the provided data. Consider, for example, a collection of data listing individuals that did or did not purchase a house from a real estate agent, where, for some reason or another, the only data remaining pertaining to these individuals is what their income level was, what their age was, and how far they have to or would have had to drive to work from their new home.

Say we get a new datapoint: income: \$25,000, age: 30, distance: 10. In this case the conditional likelihood of this data given a yes for each of the individual features is 2/9, 1/9, and 2/9 respectively. The prior on yes is 1/3. The marginal likelihoods of each the individual features are 2/9, 1/9 and 2/9 respectively. As such, the posteriors for our new datapoint are $p(y = yes|x) = \frac{(2/9)*(1/9)*(2/9)*(9/27)}{(6/27)*(3/27)*(6/27)} = \frac{0.00182}{0.00548} = 0.33$ and $p(y = no|x) = \frac{(4/18)(2/18)(4/18)(18/27)}{(6/27)(3/27)(6/27)} = \frac{0.0036}{0.00548} = 0.66$.

Note that $0.66 > 0.33$ and as such our classifier would label this datapoint with a no, this individual is not likely to purchase a house.

2.5 Support Vector Machine

The support vector machine (svm) is a two-group classification algorithm that attempts to find a hyperplane that separates the inputs within a given input space with a maximum margin of separation between the hyperplane and the “support vectors,” those vectors on either side of the hyperplane that are closest to it. To arrive at a form of the support vector machine that can be used to classify new inputs, one first needs a representation of the potential separating hyperplane

$$y_i(\mathbf{w} * \mathbf{x_i} + b) > 1$$

where y_i is the truth label of given training input x_i, w is a vector normal to our candidate separating hyperplane that represents how much “weight” is to be applied to an input, and b is a bias constant representing the threshold the weight/input product needs to pass before it is considered classified. The distance of a given hyperplane can be determined by calculating the difference between these previously mentioned “support vectors” in the direction normal to this hyperplane. This difference can be calculated via the following equation:

$$(\mathbf{x_{s+}} - \mathbf{x_{s-}})\frac{\mathbf{w}}{||\mathbf{w}||}$$

where $\mathbf{x_{s+}}$ and $\mathbf{x_{s-}}$ are respectively positive and negative support vectors and $\frac{\mathbf{w}}{||\mathbf{w}||}$ is the unit vector in the direction normal to the hyper plane towards the positive examples. The size of the margin is $2/||w||$. As such, the discovery of a working svm can be accomplished by solving a constrained optimization problem in which the thing to be minimized is $1/2||w||^2$, subject to the constraints $y_i(\mathbf{w} * \mathbf{x_i} + b) = 1$ for support vectors. Crafting an expression of this constrained optimization that can be solved by a computer can be done by taking the Lagrangian:

$$L(\mathbf{W}, , \mathbf{Y}) = 1/2||\mathbf{w}||^2 \sum_{i=1}^{n} y_i(\mathbf{w} * \mathbf{x_i} + b) - 1)$$

then taking care of the fact that the vector w_o that determines the optimal hyperplane can be written as a linear combination of the training vectors: $w_0 = \sum_{i=1}^{n} y_i \alpha_i^0 \mathbf{x_i}$ [3]. Swapping this equation into the Lagrangian yields:

$$L = \sum \alpha_i - 1/2 \sum_i \sum_j \alpha_i \alpha_j y_i y_j (\mathbf{x_i} \cdot \mathbf{x_j})$$

at which point one should consult their closest computer so that it can maximize this expression.

This final form of the Lagrangian reveals the support vector machines most powerful attribute: the kernel. The optimization of the hyperplane within the inputs depends only on their dot product of pairs of inputs; they do not appear anywhere else in the Lagrangian other than the very end and then only so as pairs of dot products.

This fact allows the writing of a decision function on new inputs:

$$f(\mathbf{u}) = \sum_{i}^{N} \alpha_i y_i (\mathbf{x_i} \cdot \mathbf{u}) + b$$

where **u** is a vector whose label we do not know. The support vector machine can use kernels to map input vectors into non-linear high-dimensional feature space without actually calculating the position of the vectors within that feature space [3]. The kernel accomplishes this by calculating the distance between (or similarity) of its two input vectors in this space without reference to their exact position within this higher space. This then allows the computation of a linear separation between the points in this higher dimensional space which translates into a non-linear separation for the vectors in their original lower dimensional space where a separation might otherwise not have been discoverable.

2.6 K-Means Clustering

The objective of the k-means algorithm is to partition a dataset into k groups such that the points within some group are all closest to the mean of that group than they are to any other group. A clear informal explanation of the work that the k-means algorithm performs was given by James MacQueen in 1967: "...the k-means procedure consists of simply starting with k groups each of which consists of a single random point, and thereafter adding each new point to the group whose mean the new point is nearest. After a point is added to a group, the mean of that group is adjusted in order to take account of the new point. Thus at each stage the k-means are, in fact, the means of the groups they represent" [4]. Formally stated, given an integer k and a set of n data points in $\mathbb{R}^d$ the K-means algorithm seeks to minimize Φ, the over all total summed in class distance between each point and its closest center such that $\lneqq = \sum_{x \in X} min_{c \in C} x - c^2$ [5].

The k-means model is a type of Gaussian mixture model that is trained with a procedure called expectation maximization. Given a set of distributions with missing data, mixture models tend to have derivatives that are either difficult to define or are entirely undefinable. On the other hand, the calculation of some ML/MAP (maximum likelihood/maximum a posteriori) estimates for some set of models can generally be calculated with little difficulty if every point within the distributions is known (at which point our learner would obviously have nothing to do) and thus calculus would be entirely unnecessary ($i.e.$, it would not matter that the derivative cannot be defined). Expectation maximization uses this fact in order to obtain an estimation of the ML/MAP indirectly. The algorithm consists of two steps. First, an estimate as to what the expected value of the hidden data is based off the current guess for the parameters is made. Then the likelihood function for the parameters is maximized under the assumption that the data discovered in the previous step is complete, $i.e.$, that there is no longer any hidden data. These steps are then repeated until some convergence criteria is met. The k-means is exactly this type of algorithm, but with the covariance matrix $\Sigma_k = \rho^2 * I_D$ and the mixing weights $\Pi_k = 1/K$ all being fixed, such that the only free parameters are the cluster centers $\mu_k \in \mathbb{R}^D$, and such that the hidden data that is the ground truth label of the data points.

2.7 Neural Networks

A neural network is a type of machine learning algorithm that mimics the interconnectivity of animal brains in order to automatically discover rules to classify given inputs. The neural network is one of the most flexible learning algorithms within literature, so flexible in fact that it is capable of approximating any continuous function [6]. As such, its inclusion within a metalearning system is almost mandatory. Genrally, a neural network system works by first being presented with a set of classified or unclassified inputs. Said system will then attempt to make a decision on these inputs on which an error value will then be assigned. The system will then see some kind of correction function applied to it. This process will continue until the system has exhausted its supply of training data, at which point it will hopefully have discovered a strong set of rules for peforming whatever work it is that it was designed to perform.

The type of neural network that will be used within this thesis is what is called the feed-forward neural network (multilayer perceptron, a.k.a. MLP). The feed forward neural network is essentially a series of logistic regression models stacked on top of each other, with the final layer being either another logicstic regression or linear regression model depending on whether or not a classification or regression problem is being solved [7]. The leftmost layer of this stack is called the input layer and consists of a set of neurons $x_i | x_1, x_2, x_3, ..., x_m$ representing the input's features. Each neuron in the hidden layer transforms the values from the previous layer via weighted linear summation $w_1 X_1 + w_2 x_2 + ... + w_m x_m$ which is then passed into a non linear-action function $g()$, such as the logistic function or the hyperbolic tangent function. It is important to note that g must be non-linear, otherwise the entire model will collapse into a large linear regression model of the form $y = w^T(Vx)$ [7].

The multi-layer perceptrons created in this experiment will be trained used in by an error propagation/training technique called backpropagation. Backpropagation is a procedure that repeatedly adjusts the weights of the connections in a neural network so as to minimize a measure of the difference between the actual output vector of the net and the desired output vector [8]. In order to accomplish this, the algorithm adjusts the weights of the nerual network by considering the error of the outputs then minimizes this error via gradient decent with respect to each of the weights within the network. Specifically, the gradient vector of the negative log likelihood error on the output neurons is computed by use of the chain rule of calculus.[7]. Say we have a one layer neural network in which the hidden layer is described by $\alpha^L = \sigma(w^L \alpha^{L-1} + b^L) = \sigma(z^L)$, where L superscript refers to the hidden layer and $L-1$ refers to the input layer of the network. The parameters of this network can be said to be $\Theta = (V, W)$ where V is the weight vector for the input layer and W is the weight vector for the hidden layer. The error (or more specifically the costs function) of a such a network is given by:

$$J(\Theta) = -\sum_n \sum_k (\hat{y}_{nk}(\Theta) - y_{nk})^2$$

in the case of regression, and via cross entropy

$$J(\Theta) = -\sum_n \sum_k y_{nk} log \hat{y}_{nk}(\Theta)$$

in the case of classification. The gradient of this error $\nabla_\Theta J$ is found via the chain rule of calculus:

$$\frac{\partial C}{\partial w^L} = \frac{\partial z^L}{\partial w^L} \frac{\partial \alpha}{\partial z^L} \frac{\partial C}{\partial \alpha^L}$$

This equation is easiest to understand if read from right to left. Notice how in each stage the rates being compared are between nearest elements, first the error to the output that produced it, then the output to the element to which the non-linearity is applied, then finally the the non-linerity recieving value to the weight vector. The result of this calculation easily gives us the direction of the gradient, the negative of which we will use to modify w^L in a direction that will reduce the output error. Reduction of the error of a multilayer perceptron with more than one neuron in each layer works mostly the same way. Once again, the chain rule of calculus is used in order to find the derivative of the output error with respect to the weights of the connections between the hidden layer before the output neurons and the output neurons $\frac{\partial C}{\partial w^L_{n-j}}$ where L is the target neurons layer (the last layer in this case), n is the index of a neuron within this layer, and j is the index of the neuron in the previous layer $L-1$ from which neuron n is recieving input (note that in this case the hyphen does not mean subtract, but rather indicates that there is a connection between these neurons). For an output neuron, its error is then given by:

$$\frac{\partial C}{\partial w^L_{n-j}} = \frac{\partial z^L_j}{\partial w^L_{n-j}} \frac{\partial \alpha^L_j}{\partial z^L_j} \frac{\partial C}{\partial \alpha^L_j}$$

with the error relative to neurons earlier in the network being calculable by continuing usage of the chain rule.

Chapter 3

Research Design

3.1 General Plan

The overall goal of this experiment is to determine whether one metalearning strategy within a given set can strictly dominate the others. The core elements required in order to determine this are: (1) a set of metalearning strategies to be compared, (2) sets of metabase datasets on which to apply the different strategies, and (3) a means to evaluate the results so as to determine the relative performance of the metalearners. The general flow of the program that constitutes this experiment begins with a set of unprocessed datasets, and then extracts the metafeatures that are required to perform dataset clustering and to run the active metalearning strategy. The program then constructs 10 metalearning bases with the elements in these sets choosen at random. A run with every machine algorithm and dataset combination is then performed, with the results being stored in the experiments database. Learning curves for each dataset/algorithm combination are then crafted. Finally, enough information now exists within the the database to run the metalearning strategies and extract results. This process is then repeated 30 times, with each run constituting an individual sample. This produces enough information in order to perform statistical analysis techniques to test the null hypothesis and obtain a margin of error of 5 percent. A detailed explanation of the experiment steps follows now, with a description of the statistical analysis to follow in Chapter 4.

3.2 Data Parsing

The data used in this experiment comes from the UCI Irvine Machine learning repository, the obtainment of which was accomplished via the use of a bash shell script that allowed the downloading of every dataset in the repository all at once. To make use of a dataset from the repository, the algorithms used in this experiment required a vector representation of the dataset currently being analyzed; the data could not be used without first translating it into this form. As such, I needed to do two things with the data before making use of them: ensure that the data could be parsed into a vector via the use of program, then write a program in order to do this. The strategy I employed in order to accomplish these goals was was twofold: I first went through the set of candidate datasets and ensured non of them was in a format so exotic that they could not be parsed programatically. Manual examination of the files revealed that those of either the .data, .svm, or .dat format were agreeable to formating and so it is these that were processed by the parser. These files were then inspected by the parser, with each dataset's column vectors being inspected one by one. Those column vectors containing only numerical data were left as is, those with any non-numerical data were assumed to be categorical, with the categories of said vector being translated to numbers with a unique number being assigned to each unique string. Rather than storing the numerical representation of each dataset within the database, the parsed form of a given dataset is crafted when it is needed, saving an enourmous amount of disk space.

3.3 Metafeature extraction

The existance of a parser allows us to craft the first table needed for this experiment which is one containing the meta-features of those datasets that are parsable. Being as how the datasets used within this project have vastly differing structures with respect to metrics such as the number of features and the maximum and minimum values of these features, the project requires a set of normalized meta-features that are applicable to any possible individual distribution or set of probability distribution(s). A set of features that meet this criteria are weighted mean, coefficient of variation, skewness, kurtosis, and entropy. The vector that represents a given dataset is crafted by taking the value of each of these attributes for each of said datasets features then normalizing them by dividing by the total number of features within that dataset, that is to say

$$F_{ad} = \frac{\sum_{c=i}^{N} f_{ai}}{N}$$

is the metafeature value a for dataset d, c is an iterator across columns for dataset d, f is the value of metafeature F applied to individual column i, and N is the number of columns within dataset d. The vector that represents a given dataset is then determined to be $V_d = (F_{1d}, F_{2d}, ..., F_{ad})$.

3.4 Metabase construction and perfomance testing

The work of a metalearning algorithm is essentially the applying the things known from a specific set of datasets towards a new, unlabeled dataset. That initial set of datasets is known as a meta database, which I shorten to metabase for convience. The core premise of this experiment is the determination as to whether or not one metalearning strategy may or may not dominate other metalearning strategies across a set of of different metabases. As such, this experiment requires multiple sets of metabases in order to produce samples that can be used to test our hypothesis. The datasets in a given metabase are randomly choosen from the set of all datasets then stored in the database. There are 10 of these per sample, each being a fifth the size of the entire set of datasets. Testing the performance of a metalearner is done by using the metalearning strategy with some given metabase and applying it to every other dataset within the set of datasets. The guesses a given metalearning strategy makes with some given metabase are then stored within a database table for later analysis.

3.5 Compiling Results

Once the guess tables are populated, it is finally possible to compile a table of results. Each entry in the table notes the metalearning strategy being evaluated, the metabase collection ($i.e.$, the sample), the metabase within that sample that the strategy used in order to analyze its test datasets, and the accuracy, training time, and rate correct score as its performance metrics, where the rate correct time measures how often the metalearner makes the correct guess given the time spent to train it in units of correct guesses per second.

Chapter 4

Research Findings

4.1 Run Results and Analysis Tools

If the no free lunch theorem applies to meta-learning strategies, we should see near equal performance across a variety of meta-set collections. We will call this assertion the null hypothesis, that is to say our null hypothesis is that the meta-learning strategies used in this experiment are equal.

In order to test the null hypothesis, 30 such samples of the kind described in Chapter 3 were collected. The metrics of interest on these samples are contained in the tables within this chapter. How often the algorithms placed first, second, or third can be seen in table 4.1. The average of these placements across all samples, i.e the algorithms average placements, can be seen in table 4.2. Table 4.3 contains the proportion of probability for the results contained in table 4.1. Table 4.4 contains the average of the proportion of probabilities contained in table 4.3 across all samples. Table 4.5 contains the standard deviations of the placement values for the meta-algorithms. Table 4.6 contains the t scores of each of the values present in table 4.1. Table 4.7 contains the average of the t scores contained in table 4.6 across all samples and it is this table that I use later to draw the conclusion that the null hypothesis may be safely rejected.

Each of the meta-set collections contains 10 basesets and the experiment compares the performance of 3 meta-learning algorithms. As such, the expected average number of first, second, and third place finishes given that the meta-learning algorithms are equal is 3.3. The values seen within the placement counts table given equal meta-learning algorithms should more often than not be either 3 or 4 and the averages of the placements across all samples should all be near 3.3. Instead, it appears that the sampler performed the best, with an average number of first place finshes of 4.5. Moreover most of the averages present in table 4.2 seem to be farther away from the expected value of 3.3 than one would intuitively expect if the meta-learning algorithms were truly equal. Whether or not these results fall far enough outside expectation in order to reject the null hypothesis requires analysis with the machinary of classical statistics. Two well established hypothesis testing measures are the method of calculating sampling distribution probabilities and t score analysis. A brief description of each of these statistical methods follows.

4.1.1 Exact Sampling Distribution

The following description losely follows the procedure described in [9]. In it, the author asks the reader to imagine testing a coin to see whether or not it is fair, flipping

	GuessesActive			GuessesEx			GuessesSamp		
	First	Second	Third	First	Second	Third	First	Second	Third
sample 1	1	4	5	6	2	2	3	4	3
sample 2	1	4	5	5	2	3	4	4	2
sample 3	1	3	6	7	3	0	2	4	4
sample 4	1	5	4	6	3	1	3	2	5
sample 5	0	6	4	8	2	0	2	2	6
sample 6	3	3	4	5	4	1	2	3	5
sample 7	4	3	3	4	4	2	2	3	5
sample 8	2	3	5	7	2	1	1	5	4
sample 9	1	3	6	3	5	2	6	2	2
sample 10	0	4	6	7	3	0	3	3	4
sample 11	0	6	4	7	3	0	3	1	6
sample 12	1	5	4	7	2	1	2	3	5
sample 13	3	3	4	5	4	1	2	3	5
sample 14	2	5	3	6	3	1	2	2	6
sample 15	2	1	7	4	6	0	4	3	3
sample 16	1	5	4	6	0	4	3	5	2
sample 17	1	4	5	6	4	0	3	2	5
sample 18	1	3	6	8	1	1	1	6	3
sample 19	1	4	5	7	3	0	2	3	5
sample 20	2	4	4	6	2	2	2	4	4
sample 21	1	2	7	4	6	0	5	2	3
sample 22	3	3	4	2	7	1	5	0	5
sample 23	3	4	3	6	4	0	1	2	7
sample 24	3	3	4	4	4	2	3	3	4
sample 25	2	6	2	7	3	0	1	1	8
sample 26	1	3	6	6	2	2	3	5	2
sample 27	7	2	1	3	5	2	0	3	7
sample 28	0	5	5	7	2	1	3	3	4
sample 29	1	2	7	4	5	1	5	3	2
sample 30	2	6	2	4	3	3	4	1	5

TABLE 4.1: Placement results

How well the meta-algorithms faired with given sample

	GuessesActive	GuessesEx	GuessesSamp
First	1.70	3.8	4.50
Second	5.57	3.3	1.13
Third	2.73	2.9	4.37

TABLE 4.2: Average placement results across all samples

the coin 1,2,..N times. He then asks the reader to consider whether some proportion of heads is actually fair from 0/N, 1/N.., N/N heads. The propability that some proportion of heads p = i/N is fair can be calculated exactly with the binomial distribution

$$\frac{N!}{i!(N-i)!}r^i(1-r)^{N-i}$$

This situation is analogous to the number of first, second, or third place finishes some meta-algorithm obtained in this thesis experiment. The probabilty of proportions for each of the meta-learning algorithms can be seen in table 4.3 and the average of these proportions across all samples can be seen in table 4.4.

We can calculate the probability of drawing either of the values closest to expectation, 3 or 4, by use of the previously mentioned binomial distribution, with $N = 10$, $r = 0.33$, and i being either 3 or 4. The values we get for the propabilities of the most expected values are then 0.26 and 0.22 respectively. The average of all values within this table is 0.15, significantly lower than the probability of the expected value. Still, this is not enough to reject the null hypothesis as proportion probability analysis does not come with a rejection criteria.

4.1.2 t score

t score analysis is a form of hypothesis testing that allows one to determine whether or not some result emerged from some given distribution via consideration of how many standard deviations the result deviates from the mean of said given distribution. Its equation has the form:

$$t = \frac{\overline{x} - \mu}{\hat{\sigma}_{\overline{x}}} = \frac{\overline{x} - \mu}{\frac{s}{\sqrt{N}}}$$

where s is the sample standard deviation, N is the number of samples, $\overline{x}$ is an individual samples mean/calculated value, and μ is the mean of the distribution of comparison.

The decision as to whether or not a specific t score value implies a result is from a different distribution depends on how many samples were used in the calculation of the sample mean and on what the desired confidence interval is. The critical threshold used in order to make this decision is gained by the use of t distribution table. In the case where 30 samples are used and the desired margin of error is 5%, the critical thresholds for a two tailed t test are -2.042 and 2.042. If the averaged values of the t scores falls outside of these bounds then we can reject the null hypothesis with a 5 % margin of error. The standard deviations and t scores for each of the samples can be seen in table 4.4. Taking the average of the absolute value of each of these t scores yeilds 5.11. We can thus comfortably reject the null hypothesis.

	GuessesActive			GuessesEx			GuessesSamp		
	First	Second	Third	First	Second	Third	First	Second	Third
sample 1	0.09	0.23	0.14	0.06	0.20	0.20	0.26	0.23	0.26
sample 2	0.09	0.23	0.14	0.14	0.20	0.26	0.23	0.23	0.20
sample 3	0.09	0.26	0.06	0.02	0.26	0.02	0.20	0.23	0.23
sample 4	0.09	0.14	0.23	0.06	0.26	0.09	0.26	0.20	0.14
sample 5	0.02	0.06	0.23	0.00	0.20	0.02	0.20	0.20	0.06
sample 6	0.26	0.26	0.23	0.14	0.23	0.09	0.20	0.26	0.14
sample 7	0.23	0.26	0.26	0.23	0.23	0.20	0.20	0.26	0.14
sample 8	0.20	0.26	0.14	0.02	0.20	0.09	0.09	0.14	0.23
sample 9	0.09	0.26	0.06	0.26	0.14	0.20	0.06	0.20	0.20
sample 10	0.02	0.23	0.06	0.02	0.26	0.02	0.26	0.26	0.23
sample 11	0.02	0.06	0.23	0.02	0.26	0.02	0.26	0.09	0.06
sample 12	0.09	0.14	0.23	0.02	0.20	0.09	0.20	0.26	0.14
sample 13	0.26	0.26	0.23	0.14	0.23	0.09	0.20	0.26	0.14
sample 14	0.20	0.14	0.26	0.06	0.26	0.09	0.20	0.20	0.06
sample 15	0.20	0.09	0.02	0.23	0.06	0.02	0.23	0.26	0.26
sample 16	0.09	0.14	0.23	0.06	0.02	0.23	0.26	0.14	0.20
sample 17	0.09	0.23	0.14	0.06	0.23	0.02	0.26	0.20	0.14
sample 18	0.09	0.26	0.06	0.00	0.09	0.09	0.09	0.06	0.26
sample 19	0.09	0.23	0.14	0.02	0.26	0.02	0.20	0.26	0.14
sample 20	0.20	0.23	0.23	0.06	0.20	0.20	0.20	0.23	0.23
sample 21	0.09	0.20	0.02	0.23	0.06	0.02	0.14	0.20	0.26
sample 22	0.26	0.26	0.23	0.20	0.02	0.09	0.14	0.02	0.14
sample 23	0.26	0.23	0.26	0.06	0.23	0.02	0.09	0.20	0.02
sample 24	0.26	0.26	0.23	0.23	0.23	0.20	0.26	0.26	0.23
sample 25	0.20	0.06	0.20	0.02	0.26	0.02	0.09	0.09	0.00
sample 26	0.09	0.26	0.06	0.06	0.20	0.20	0.26	0.14	0.20
sample 27	0.02	0.20	0.09	0.26	0.14	0.20	0.02	0.26	0.02
sample 28	0.02	0.14	0.14	0.02	0.20	0.09	0.26	0.26	0.23
sample 29	0.09	0.20	0.02	0.23	0.14	0.09	0.14	0.26	0.20
sample 30	0.20	0.06	0.20	0.23	0.26	0.26	0.23	0.09	0.14

TABLE 4.3: Placement results proportion probabilities

Proportion probabilities of placement results if the null hypothesis were true

	GuessesActive	GuessesEx	GuessesSamp
First	0.13	0.19	0.16
Second	0.10	0.19	0.11
Third	0.19	0.20	0.16

TABLE 4.4: Average of proportion probabilities across all samples

	GuessesActive	GuessesEx	GuessesSamp
First	1.42	1.30	1.48
Second	1.54	1.53	1.06
Third	1.36	1.33	1.58

TABLE 4.5: Placement results standard deviations across all samples

algorithms	GuessesActive			GuessesEx			GuessesSamp		
positions	First	Second	Third	First	Second	Third	First	Second	Third
sample 1	-10.41	3.24	7.13	10.93	-5.51	-7.98	-1.54	3.18	-1.33
sample 2	-10.41	3.24	7.13	6.83	-5.51	-2.00	3.09	3.18	-5.33
sample 3	-10.41	-1.62	11.41	15.04	-1.38	-19.96	-6.18	3.18	2.67
sample 4	-10.41	8.10	2.85	10.93	-1.38	-13.97	-1.54	-6.36	6.67
sample 5	-14.87	12.96	2.85	19.14	-5.51	-19.96	-6.18	-6.36	10.67
sample 6	-1.49	-1.62	2.85	6.83	2.75	-13.97	-6.18	-1.59	6.67
sample 7	2.97	-1.62	-1.43	2.73	2.75	-7.98	-6.18	-1.59	6.67
sample 8	-5.95	-1.62	7.13	15.04	-5.51	-13.97	-10.81	7.95	2.67
sample 9	-10.41	-1.62	11.41	-1.37	6.89	-7.98	12.36	-6.36	-5.33
sample 10	-14.87	3.24	11.41	15.04	-1.38	-19.96	-1.54	-1.59	2.67
sample 11	-14.87	12.96	2.85	15.04	-1.38	-19.96	-1.54	-11.13	10.67
sample 12	-10.41	8.10	2.85	15.04	-5.51	-13.97	-6.18	-1.59	6.67
sample 13	-1.49	-1.62	2.85	6.83	2.75	-13.97	-6.18	-1.59	6.67
sample 14	-5.95	8.10	-1.43	10.93	-1.38	-13.97	-6.18	-6.36	10.67
sample 15	-5.95	-11.34	15.69	2.73	11.02	-19.96	3.09	-1.59	-1.33
sample 16	-10.41	8.10	2.85	10.93	-13.77	3.99	-1.54	7.95	-5.33
sample 17	-10.41	3.24	7.13	10.93	2.75	-19.96	-1.54	-6.36	6.67
sample 18	-10.41	-1.62	11.41	19.14	-9.64	-13.97	-10.81	12.72	-1.33
sample 19	-10.41	3.24	7.13	15.04	-1.38	-19.96	-6.18	-1.59	6.67
sample 20	-5.95	3.24	2.85	10.93	-5.51	-7.98	-6.18	3.18	2.67
sample 21	-10.41	-6.48	15.69	2.73	11.02	-19.96	7.72	-6.36	-1.33
sample 22	-1.49	-1.62	2.85	-5.47	15.15	-13.97	7.72	-15.91	6.67
sample 23	-1.49	3.24	-1.43	10.93	2.75	-19.96	-10.81	-6.36	14.67
sample 24	-1.49	-1.62	2.85	2.73	2.75	-7.98	-1.54	-1.59	2.67
sample 25	-5.95	12.96	-5.71	15.04	-1.38	-19.96	-10.81	-11.13	18.67
sample 26	-10.41	-1.62	11.41	10.93	-5.51	-7.98	-1.54	7.95	-5.33
sample 27	16.36	-6.48	-9.99	-1.37	6.89	-7.98	-15.45	-1.59	14.67
sample 28	-14.87	8.10	7.13	15.04	-5.51	-13.97	-1.54	-1.59	2.67
sample 29	-10.41	-6.48	15.69	2.73	6.89	-13.97	7.72	-1.59	-5.33
sample 30	-5.95	12.96	-5.71	2.73	-1.38	-2.00	3.09	-11.13	6.67

TABLE 4.6: Placement results t scores

t score for each individual placement result

	GuessesActive	GuessesEx	GuessesSamp
First	-7.29	2.27	4.99
Second	9.16	-0.14	-13.17
Third	-2.78	-2.07	4.13

TABLE 4.7: Average of t scores across all samples

Chapter 5

Conclusion and Future Work

In this thesis, I proposed that the no free lunch hypotheis might not apply to meta learning algorithms. In order to test this hypothesis, I first built a system to determine the accuracy of three meta learning strategies: Exhaustive, Active, and Sampling. To use these strategies, a base of datasets would first be randomly choosen from the collection of all available datasets that had been gathered from the UCI Irvine data repository. Each strategy would then be carried out on the metabase and make an estimate as to what algorithm would result in the highest classification accuracy. Each algorithm would make this guess for each dataset in the collection of available datasets excluding the datasets within the current metabase. A new metabase would then be choosen at random and the process would be repeated 9 more times, giving a number between 0 and 10 for how many times each algorithm got the First, second, or third most correct guesses. This process was repeated 30 times, resulting in 30 samples. t test analysis was then performed, giving an an average among the absolute values of each of the position t scores of 5.11, allowing us to reject the null hypothesis at a 5 percent margin of error.

A flaw exist within this experiment that I would correct had I more time: I was only able to obtain one set of datasets with 88 instances in it. As such there is a possibility of bias in the data. This potential bias could be mitigated with the introduction of extra sets of datasets, with the ideal being a unique set of datasets for each sample.

Bibliography

[1] T. Bhatt and G. Bhatt, “Ranking of classifiers based on dataset characteristics using active meta learning”, *International Journal of Computer Applications*, vol. 69, no. 20, pp. 31–36, 2013. [Online]. Available: https://pdfs.semanticscholar.org/93ed/31bf211362c7b77f254d0fb8d5f2d1a101a8.pdf.

[2] R. Leite and P. Brazdil, “Predicting relative performance of classifiers from samples”, *ICML 05 Preceedings of the 22nd International conference on Machine learning*, pp. 497–503, Aug. 2005. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.74.641&rep=rep1&type=pdf.

[3] V. Vapnik and C. Cortes, “Support-vector networks”, *Machine Learning*, vol. 20, no. 3, pp. 273–297, Sep. 1995. [Online]. Available: https://link.springer.com/article/10.1007/BF00994018.

[4] J. MacQueen, “Some methods for classification and analysis of multivariate observations”, *Fifth Berkeley Symp on Math Statistics and Probability*, vol. 1, pp. 281–197, 1967. [Online]. Available: https://projecteuclid.org/euclid.bsmsp/1200512992.

[5] D. Arthur and S. Vassilvitskii, “K-means ++: The advantages of careful seeding”, *SODA 07 Proceedings of the eighteenth annual ACM-SIAM symposium on discrete algorithms*, pp. 1027–1035, Jan. 2007. [Online]. Available: http://ilpubs.stanford.edu:8090/778/1/2006-13.pdf.

[6] K. Hornik, “Approximation capabilities of multilayer feedforward networks”, *Neural Networks*, vol. 4, pp. 251–257, 1991. [Online]. Available: http://zmjones.com/static/statistical-learning/hornik-nn-1991.pdf.

[7] K. P. Murphy, *Machine Learning: A probabilistic Perspective*.

[8] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning representations by back-propagating errors”, *Nature*, vol. 323, pp. 533–536, Oct. 1986.

[9] P. R. Cohen, *Empirical Methods for Artificial Intelligence*.