DELAWARE STATE UNIVERSITY

MASTERS THESIS

A comparison of metalearning strategies

Author: *Supervisor:*

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

in the

CIBIL
Department of computer science

October 15, 2019

Declaration of Authorship

I, John LIDDELL, declare that this thesis titled, "A comparison of metalearning strategies" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: _____

Date: _____

Acknowledgements

The acknowledgments and the people to thank go here, don’t forget to include your project advisor…

Contents

Declaration of Authorship iii

Acknowledgements v

1 Introduction 3

2 Review of the literature 5
2.1 No Free lunch theorem . . . 5
2.1.1 Brute force metabase . . . 5
2.1.2 Active Meta Learning . . . 5
2.1.3 Predicting relative performance of classifiers from samples . 6
2.2 Summary of producable machines . . . 6
2.3 Linear Regression . . . 6
2.4 Naive Bayes . . . 7
2.5 Support Vector Machine . . . 8
2.6 K-Means clustering . . . 9
2.7 Neural Networks . . . 10

3 Research Design 13
3.1 General Plan . . . 13
3.2 Data Parsing . . . 13
3.3 Metafeature extraction . . . 14
3.4 Metabases Construction and perfomance testing . . . 14
3.5 Compiling Results . . . 14

4 Research Findings 15
4.1 Run results and Analysis tools . . . 15
4.1.1 Exact Sampling Distribution . . . 15

A Frequently Asked Questions 17
A.1 How do I change the colors of links? . . . 17

List of Figures

List of Tables

4.1 Averaged t scores 15

List of Abbreviations

LAH	**L**ist **A**bbreviations **H**ere
WSF	**W**hat (it) **S**tands **F**or

Physical Constants

Speed of Light $c_0 = 2.997\,924\,58 \times 10^8\,\mathrm{m\,s^{-1}}$ (exact)

List of Symbols

a	distance	m
P	power	W (J s^{-1})
ω	angular frequency	rad

For/Dedicated to/To my…

DELAWARE STATE UNIVERSITY

Abstract

Faculty Name
Department of computer science

Masters in Science

A comparison of metalearning strategies

by John LIDDELL

A comparison of several metalearning strategies in order to ascertain the truth of the no free lunch theorem as applies to metalearning strategies is performed. The information needed in order to validate the performance of each metalearning strategy was gathered and stored in a database. Each metalearning strategy was then compared in terms of their accuracies, training times, and rate correct scores across the various metabases stored within the database.

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 performance of all machine learning algorithms even out across all datasets, that is to say that no one machine learning algorithm is best in every situation, performance is contengent on the problem space in which the algorithm is operating. As such, the decision of algorithm is non 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 neccessary to acquire access to such an expert, in which case the usage of a metalearner becomes appropriate. With a metalearner one feeds the metalearner 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 metalearner itself must first be trained, and this training itself requires some sort of learning strategy. This fact suggests thath the decision of what metelearning strategy to use for some given body of datasets should be succeptable to the previously mentioned no free lunch theorem, that is to say that some metalearning 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 metalearning strategy strictly dominates. Chapter 5 concludes with an acceptance or rejection of the stated hypothesis and proposes possible followups this experiment.

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 perfomance 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 observation 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 distriubtion 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 from 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 involves the collecting of the run statistics of the set of machines that a given meta learner can produce applied to a metabase with a clustering algorithm to produce results with new datasets. This version of metalearning will act as a sort of control for this study, the results of the more complicated meta learning strategies that follow only really mean anything with respect to how long it would've taken to train and how long it would've performed if we had just trained it via brute force.

2.1.2 Active Meta Learning

The first of the metalearning 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" **Bhatt**. What this entails is a decision on what datasets to allow into a metalearners metabase, rather than analyze every candidate meta base dataset an active metalearner will analyze the next dataset with the highest uncertainty. The relative uncertainty between two datasets is defined by

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

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 corrosponding minimum. Determining which dataset has the overall highest uncertainty can be done by summing over the relative values for each metaparameter, ranking them then collecting their overall uncertainty scores then choosing the one that is highest where

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

is the equation representing the overall uncertainty for dataset d_i in metaparameter V_x.

2.1.3 Predicting relative performance of classifiers from samples

The second of the metalearning strategies implemented within this study is one in which a representive subsection of the metabase is trained with each algorithm entirely, after which point the rest of the metabase under goes a sort of curve sampling analysis in which the results of future learning is predicted based off what dataset the new datasets learning curve most resembles. **Leite** As with the other two meta learning strategies, the label for new datsets is the determined via clustering with the datasets within this post trained metabase.

2.2 Summary of producable machines

The strategies mentioned in the previous section must be able to produce learning algorithms to be considered metalearing machines. The machines that these strategies can produce are the K-means clusterer, 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 earning 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 within literature. It asserts that the response is a linear function of the inputs. **murphy**. 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

$$S(\beta) = (y_1 - \beta x_1)^2 + (y_2 - \beta x_2)^2 + ...(y_3 - \beta x_3)$$

then take the partial deriviate of this sum of sqaured deviations with respect to each of the components of β, set them to zero, then solve the resulting equations to directly determine what the values of the parameters are that minimizes the sum of the squared errors of the system. With linear regression in two dimensions (one dimension in the independant variable and one dimension in the dependant 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 independance. 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 independance easily 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 so "by hand" or can be done so from distributions that are infered 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 to drive to work from their new home. A set of possible example points is as follows:

Say we get a new datapoint: $25, 0003010$. 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.

This "by hand" calculation of the posterior for every new data point is fine when the number of examples is low but can quickly grow out of hand when the number of data points is very large. In this case it is better to extract the parameters of the assumed distributions of the various portions of the equation (for the data likelihoods, the prior, the marginals, etc). For this example the data likelihoods and marginals are all gaussians. The relative means of each of the distributions can easily be seen from the data table (35,000, 40, 15 for each of the features marginal and conditionl

probabilities). The variances can be calculated by summing the distances from the means and dividing by the number of points in each given distribution, at which point all the means and variances will be known. Plugging this values into the definition of the gaussian

$$p(x|\mu, \rho^2) = \frac{1}{\sqrt{2\pi\rho^2}} e^{\frac{x-\mu^2}{\rho^2}}$$

would give us the probability for each of the values for a given new data point, at which point the calculation of the posterior would follow in exactly the same fashion as the "by hand" example.

2.5 Support Vector Machine

The support vector machine is a two-group classification algorithm that attempts to find a hyper plane that seperates the inputs within the given input space with a maximum margin of seperation between the hyper plane and the "support vectors", those vectors on either side of the hyper plane that are closest too 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 seperating hyper plane

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

such that y_i is the truth label of given training input x_i, w is a vector normal to our candidate seperating hyper plane 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 given hyper plane can be determined by calculating the difference between these previously mentioned "support vectors" in the direction normal to this hyper plane. This difference can be calculated via

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

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 found 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: $w_0 = \sum_{i=1}^{n} y_i \alpha_i^0 \mathbf{x_i}$ **Vapnik**. 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 who's label we do not know. The supporector machine can use "kernels", to map input vectors into non-linear high-demensional feature space without actually calculating the position of the vectors within that feature space **Vapnik**. 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 seperation between the points in this higher dimensional space which translates into a non-linear seperation for the vectors in their original lower dimensional space where a seperation might other wise not have been discoverable.

2.6 K-Means clustering

The objective of the k-means algorithm (which results in a k-means model) 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 algorith performs was given by James McQueen 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." **MacQueen** 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 $\lesseqgtr = \sum_{x \in X} min_{c \in C} x - c^2$ **Arthur**.

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 derivitaves that are either difficult to define or entirely undefinable. On the other hand, the calculation of some ML/MAP 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 unneccessary (i.e it wouldn't matter that the derivitive cant be defined). Expectation maximization uses this fact in order to obtain an estimation of the ML/MAP in a roundabout way.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 convergance 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. Being that it is one of the most flexible learning algorithms within literature (actually able to approximate any continuous function)**Hornik**, its inclusion within a metalearning system is almost mandatory. Genrally, such a system works by first being presented with a set of classified or unclassified inputs. Said neural network 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 aka 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.**Murphy**. 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 $w_1X_1 + w_2x_2 + ...w_mx_m$ 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 $y = w^T(Vx)$. **Murphy**

In order to train the MLP it provides, sci-kit learn uses 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. **Rumelhart**. 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.**Murphy**.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 their 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. *Place a figure here* . For an excellent and intuitive explanation of how neural networks work, pls consider viewing the animated overview of the method at 3Blue1Brown's youtube channel.

Chapter 3

Research Design

3.1 General Plan

The overall goal of this experiment is to determine whether or not one metalearning strategy within a given set can strictly dominate the others. The core elements required in order to determine this are a set of metalearning strategies to be compared, sets of metabase datasets on which to apply the different strategies, and a means to evaluate the results so as to determine the relative performance of the metalearners. The general flow of the program tha constitutes this experiment begins with a set of unprocessed datasets 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 runs all table. 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, at which point statistical analysis techniques can be used to test the null hyptothesis. A detailed explanation of each of these steps follows.

3.2 Data Parsing

All machine learning algorithms boil down to a set of mathematical equations at the end of the day, with some vector representing a dataset fed in at one end and a number representing either a catagory or quantity being expelled at the other. As such any metalearning system must have with it a means to parse the datasets it means to process. The strategy I employed in order to ensure usable data was two fold: I first went thru 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 are processed and used by the parser. These files are then inspected by the parser, with each datasets column vectors being inspected one by one. Those column vectors containing only numerical data are left as is, those with any non-numerical data are 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 disk space and making the project far easier to understand.

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 along lines 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, skew, 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 Metabases 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 expereiment requires multiple metabases in order to produce results 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, 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 set 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

In order to test the null hypothesis, 30 such samples of the kind described in chapter 3 were collected. The content of the samples appear in the following tables.

These results are clearly not what we expect if the null hypothesis is true. Testing of this hypothesis can be done with the machinary of classical statistics, specifically by calculating the exact probability of the results by working out the exact sampling distribution and by obtaining how likely these results are to come from the various appropriate distributions where the performances are equal by running the results thru the machinary of the t-test. A description of each of these statistical methods follows.

4.1.1 Exact Sampling Distribution

The following description losely follows the procedure described in **Cohen**. In it, the author asks you to imagine testing a coin to see whether or not it is fair, flipping the coin 1,2,..N times. He then asks you 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 proportion for each of the meta-learning algorithms can be seen in table

T test equation where s is the sample standard deviation and N is the number of samples, overscore x is an individual samples mean/calculated value, mu is the population mean/expected value

	GuessesActive	GuessesEx	GuessesSamp
0	-7.2863	9.15791	-2.78077
1	2.26812	-0.137718	-2.06779
2	4.99363	-13.1737	4.13425

TABLE 4.1: Averaged t scores

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

Appendix A

Frequently Asked Questions

A.1 How do I change the colors of links?

The color of links can be changed to your liking using:

`\hypersetup{urlcolor=red}`, or
`\hypersetup{citecolor=green}`, or
`\hypersetup{allcolor=blue}`.

If you want to completely hide the links, you can use:

`\hypersetup{allcolors=.}`, or even better:
`\hypersetup{hidelinks}`.

If you want to have obvious links in the PDF but not the printed text, use:

`\hypersetup{colorlinks=false}`.