In this thesis, I conduct an experiment to see whether or not the No free lunch theorem applies to meta-learning strategies.

When analyzing a dataset, one must decide on an algorithm. This decision is very often made by an individual with both familiarity in the datasets relevant problem domain and in machine learning literature. The deciding of an algorithm by such an expert is, however, not always necessary. Meta-learning strategies automate the algorithm selection process by use of stored run statistics for previously analyzed datasets. Given a new dataset a meta-learning algorithm first uses some measure of similarity between the new dataset and those datasets that had previously been analyzed, hereafter referred to by the term "metabase". After selecting the dataset of greatest similarity from the metabase, the meta-learner will return the algorithm that maximizes the desired run statistic, which in the case of this experiment,is the classification accuracy.

The reason a selection must be made on an algorithm at all is that the performance an algorithm has on a dataset is contingent on what problem that dataset extends from. This phenomenon was first formally noted by Wolpert and Macready in "No free lunch theorems for optimization". In this paper they demonstrate that an algorithm that has enhanced performance in one specific problem domain will have as a consequence of this enhanced performance degraded performance in other problem domains. As such, the decision of what algorithm to use in order to analyze a dataset is non-arbitrary; no one algorithm is best at analyzing every dataset.

The decision as to what algorithm to use on a dataset itself constitutes a sort of learning algorithm, with different individuals having different rules behind their decision of algorithm choice and different meta-learning systems employing different strategies in order to automate this decision. This implies that the no free lunch theorem should apply to meta-learning strategies as well, that is to say no one meta-learning strategy should perform best with every possible metabase.

As such the goal of this experiment was to ascertain whether or not the no free lunch hypothesis applies to meta-learning algorithms, that is to say to ascertain whether or not some meta-learning algorithms strictly dominate other meta-learning algorithms. A few terms from literature must be defined before continuing on to a description of the methodology.

The three meta-learning algorithms that I chose for comparison were the Brute Force Algorithm, the Active Meta-learning strategy and the Strategy of Nearest Learning Curve Comparison. Each of these algorithms takes as its inputs a metabase and a dataset outside of that metabase and produces a guess at what algorithm would best classify this new dataset.

The Brute Force approach to meta-learning takes a given metabase as is, then when given a new dataset applies a given similarity measure (in the case of this thesis the clustering algorithm) then returns the algorithm that performed best on the dataset from the metabase that was found to be most similar to the new dataset.

Introduced in the journal article "Ranking of Classifiers based on Dataset Characteristics using Active Meta Learning", Active Meta Learning is a learning

strategy that reduces the cost of generating meta examples by selecting relevant data.

It is a meta-learning technique in which a candidate dataset is only allowed into the metabase if it has a higher uncertainty score relative to its peers. 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 dataset that has the highest score 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.

The paper states that this process trains its meta-learner in less time than a brute force learner and that the resultant meta-learner also has a higher classification accuracy than a brute force learner. Moreover, due to it having a smaller metabase, the time it takes for a system that trains its meta-learner with an active learner will also take less time to classify new datasets than a brute force learner.

Introduced in the paper "Predicting Relative Performance of Classifiers from Samples", the strategy of nearest learning curve comparison gathers run statistics on the datasets within its metabase at various fractions of the full size of the training portion of that dataset. It is then possible to plot a 2D learning curve, with the x axis of such a graph being a specific fraction of the training set and the y axis of such a graph being the test classification accuracy at that percent.

The categorization of new datasets can then be accomplished in two steps.

First, a model is trained on the new dataset with some fractions of its training set using each candidate algorithm. These results are then compared with the learning curves of each of the sets in the metabase. The meta-learner then returns the best algorithm of the dataset within the metabase with which the new datasets learning curves is most similar. The authors state that this process trains its meta-learner in less time than a brute force learner and the resultant meta-learner will also have a higher classification accuracy than a brute force meta-learner.

The machine-learning algorithms that these meta-machine learning strategies can return as their answers are one of either K-means clustering, neural network, Naive Bayes classifier, support vector machine or regression.

Linear regression is a method that asserts that the response is a linear function of the inputs. 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.

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 this assumption of independence, generate a guess as to 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 support vector machine is a two-group classification algorithm that attempts to find a hyperplane that separates the inputs within the 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.

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 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."

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 the inputs given to it. A neural network system takes the inputs given to it, attempts to make a decision on these inputs, then assigns, an error value to the decision it made on these inputs based off how close the answer produced by the system was to its true value. The system then uses a correction function, oftentimes backpropagation, in order to modify the weights of the connections between the neurons within the network. This process will continue until the system has obtained the desired error within its supply of training data or until it reaches a point where iterations of this improvement cycle no longer results in improved performance.

The overall goal of this experiment was to determine whether or not one meta-learning strategy within a given set can strictly dominate the others.

The elements required in order to determine this were a set of meta-learning strategies to be compared, a pool of datasets from which draw metabases, gather

run statistics, and compare meta learner performances, and a means to evaluate the results so as to determine the relative performance of the meta learners.

I wrote a program in order to meet these requirements. An overview of the procedure this program goes through now follows

The program first begins with a set of unprocessed datasets, then extracts the metafeatures that are required to perform dataset clustering and to run the active meta-learning strategy, storing the results in a local database table. On completion of this task the program then iterates thru each of the datasets in our available pool of datasets and extracts run statistics for each of these datasets; for each available base algorithm a model is trained and classification is performed on each instance in the dataset that was not used for the training. The time taken to train the model and the accuracy of the trained model are then stored within the database. On completion of this task the program then once again iterates through the datasets but this time collects learning curves of the type described earlier, storing them within their own table in the local database. On completion of this task, the program then goes on to construct 30 collections of the sets of the metabases that will be needed in order to produce the samples that will be needed for later analysis. On completion of this task, the system then applies each of the meta-learning strategies to each of the metabase sets within each of the metabase set collections, storing the guesses made by a meta-learner within individual database tables. The program finishes by first compiling the result table containing the relative placement counts of the meta learners, that is to say how well a meta learner did with some specific meta base relative to its neighboors, then produces the neccessary charts needed to perform analysis. A more detailed explanation of each of the steps taken by this experiments program will follow after a quick description of the development environment.

The languages of use for the experiments program and its utilities were python 3.7 and bash, with bash having been used for the script that gathered the datasets and python having been used for everything else. The editors used to develop the code were emacs and pycharm; primarily emacs at first with pycharm later having been added for its powerfull linter and analysis tools. The experiment was tested and run via the use of ipython in a windows scripting environment called powershell. The computer on which the experiment was run was my personal pc. Its metrics are as follows: RAM: 16 GB, Processor: Intel i5-4460, Operating System: Windows 10.

The data used in this experiment was gathered programmatically from the UCI Irvine Machine Learning Repository. In order to ensure the data could be used by the experiments program, I iterated thru a cycle in which I wrote parsing code in the parsing portion of the experiments program, tested the results, then manually examined the data I was attempting to parse to determine why I could not parse it if I could not parse it. Many iterations of this procedure revealed that data files 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.

In order to make a file available to the system, the parser first checks to ensure it is one of the allowed types. After confirming this, it observes the

columns of the imported data to ascertain what type of data is contained within those columns. If a column contains anything other than numerical data, the parser numbers each unique entry in that column then replaces the entry with its number.

The resulting matrix representations of the data files can thus be used for classification by the base machine learning algorithms, but were too large and unwieldy for inclusion in a database. As such, only a small meta vector feature representation of the data was stored within the database. Instead of permanant storage, the datasets would be parsed only when needed.

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 required 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 met this criteria were 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 $V_d = (F_{1d}, F_{2d}, ...F_{ad})$. It is this vector that is then stored within the database for later use.

A quick description of the experiments database solution is appropriate here for clarity. The experiments database is created programmatically. A type of software known as an "Object Relational Mapper", a program that can map data types to database statements, was used to create and maintain this database. The name of the python module used in the experiments program for this purpose is sqlalchemy. Contained within the database were tables to store algorithm information, dataset information, algorithm run results, learning curves, base set collections, guess tables, and meta algorithm run results.

Rather than run a selected algorithm for testing when selected by a meta algorithm, every algorithm and dataset combination is trained once, with the results of the analysis session then being stored within the database. An analysis session would go as such:

First, the target dataset is parsed and the order of the rows in the matrix returned from this parsing procedure would be randomized. The first 20 percent of this randomized dataset is then chosen as the training set and the later 80 percent as the testing set. For each algorithm in the system, a model is then trained with the training set and tested with the testing set. The versions of the algorithms used in this system are provided via a python machine learning library called "sci-kit learn", which is a machine learning extension to the SciPy

ecosystem. The time it took to train the model and its labeling accuracy are then stored within the database.

This same procedure is used to gather and store learning curves within the database; but in the instance of the learning curve, 3 different models would be trained for each algorithm: 1 with 10 percent of the training set, 1 with 20 percent of the training set, and 1 with 30 percent of the training set. The accuracies of these models would then be stored within the database.

Metabase collections are then created using the pool of datasets available within the datasets table. Each metabase collection contains within it 10 metabase sets and each metabase set contains within it 10 datasets. The datasets chosen for inclusion within the metabases are randomly chosen from the pool of available datasets. 30 such metabase collections are gathered, with each metabase collection stored within its own database table.

The meta-learning algorithms used within this experiment were all written by me using the descriptions of the algorithms available within the papers in which they were the topic. I would have preferred to use instances of the algorithms available from the authors themselves but I was unsuccessful in obtaining them.

The meta-learners were then tested against each metabase collection. The procedure went as follows: A metabase collection is selected. A metabase within this metabase collection is then selected. A meta-algorithm is then selected. The meta-algorithm uses its learning strategy to train with the metabase. The trained meta-learner then makes a guess as to what it thinks the best algorithm is to train each of the other sets not currently within its metabase. The guess made by the algorithm, the correct best performing dataset, whether or not the guess was correct, what metabase was used to make the guess and what collection that metabase exists within are then all stored within the database. This process is repeated for every combination of meta algorithm, metabase collection and metabase within the collections.

The last table required within the database for analysis to become possible is the results table. Each results table row contains the name of a meta algorithm and what its testing accuracy and training time was when trained with a meta base from a given meta collection.

With the results now available within the database, analysis is now possible. The system iterates thru each metabase collection name, crafting a results matrix for each collection from the results present in the results database table. Each row of the matrix represents a meta-learning algorithm, and each column within this matrix represents how many times that algorithm got that relative results placement within the metabases that comprises that collection.

Say, for instance, the active meta learner had the highest accuracy when using 1 of the metabases within the first collection, the second highest accuracy 4 times when using the metabases within the first collection, and the worst accuracy 5 times when using the metabases within the first collection. The row representing its result is thusd 1 4 5, as can be seen here in the chart. Note that because there are 10 metabases within each meta-collection, the sum of each row is always 10. A matrix of this sort is created for each metabase collection.

In order to test the null hypothesis, 30 such samples of the kind previously described were created.

If each of the meta learning algorithms were truly equal, we would expect the averaged numbers for each of the positions to be near 3.3. Instead, it appears that the sampler performed the best, with an average number of first place finishes of 4.5 and the active strategy performed the worst, with an average number of first place finishes of 1.7. Whether or not these results fall far enough outside expectation in order to reject the null hypothesis requires the machinary of classical statistics. A common and reliable way of measuring how unlikely a set of results is is the t test, and it is this test that is used to test these results.

The t test equation is as follows:

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

where s is the sample standard deviation, N is the number of samples, $\overline{x}$ is the sample mean, and μ is the population mean/expected value.

In this equation, the diffrence between the observed mean of the sample and the expected mean is taken and the result is normalize, giving the number of sample standard deviations by which the observed sample distributions mean differs from expectation. The fewer the number of samples, the more the number of standard deviations the means need to differ for rejection at some desired margin of error. In the case where we have 30 samples, the critical thresholds for a two tailed test (a test in which positive and negative differences are considered noteworthy) at a 5 percent margin of error are -2.04 and 2.04.

If the averaged values of the t scores falls outside these bounds then we can reject the null hypothesis with a 5 percent margin of error. The standard deviations and t scores for each of the samples can be seen in the following table.

In order to determine whether or not the results as a whole fall significantly outside of expectation we can take the average of all the averaged t scores absolute values. This will give as a measure of how much the values observed her deviate from expectation as a whole. The value we get after under taking this procedure is 4.42, more than twice the t score value needed in order to reject the null hypothesis at 5 percent margin of error. We can thus comfortably reject the null hypothesis.

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 our available pool of datasets. Each strategy would then act on the metabase in its own fashion 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 pool excluding the ones in the current base. A new base would then be choosen and the process repeated 9 more times, giving a number between 0 and 10 for how many times each algorithm got the most/second most/least number of correct guesses. This

process was repeated 30 times, giving us 30 samples. Statistical analysis was then performed, giving an average among the absolute values of each of the t scores of 5.11, allowing us to reject the null hypothesis at a 5 percent margin of error.

As a followup to this experiment, I would have liked to repeat the process with more datasets. Given that the size of the pool of available datasets is 88 there exists a chance of bias in the metabase sets. Increasing the size of the pool of datasets to increase the uniqueness of each of the metabases would reduce this bias likelihood. Idealistically, each of the datasets in each of the metabases would be unique. This would require 30 * 100 datasets = 3000 observed, programmatically parsable datasets, a number which is prohibitively difficult to acquire. If, however, this number of datasets becomes aquirable in the future, this experiment would be worth repeating with this new body of datasets.