Meta Learning Recommendation System for Classification

THESIS

Clarence O. Williams III, 1st Lieutenant, USAF

AFIT-ENS-MS-20-M-181

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT-ENS-MS-20-M-181

META LEARNING RECOMMENDATION SYSTEM FOR CLASSIFICATION

THESIS

Presented to the Faculty

Department of Operational Sciences

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

in Partial Fulfillment of the Requirements for the

Degree of Master of Science in Operations Research

Clarence O. Williams III, BS

1st Lieutenant, USAF

March 2020

AFIT-ENS-MS-20-M-181

META LEARNING RECOMMENDATION SYSTEM FOR CLASSIFICATION

THESIS

Clarence O. Williams III, BS
1st Lieutenant, USAF

Committee Membership:

Dr. J. D. Weir, PhD
Chairman

Capt Phillip R. Jenkins, PhD
Member

AFIT-ENS-MS-20-M-181

Abstract

A data driven approach is an emerging paradigm for the handling of analytic problems. In this paradigm the mantra is to let the data speak freely. However, when using machine learning algorithms, the data does not naturally reveal the best or even a good approach for algorithm choice. One method to let the algorithm reveal itself is through the use of Meta Learning, which uses the features of a dataset to determine a useful model to represent the entire dataset. This research proposes an improvement on the meta-model recommendation system by adding classification problems to the candidate problem space with appropriate evaluation metrics for these additional problems. This research predicts the relative performance of six machine learning algorithms using support vector regression with a radial basis function as the meta learner. Six sets of data of various complexity are explored using this recommendation system and at its best, the system recommends the best algorithm 67% of the time and a "good" algorithm from 67% to 100% of the time depending on how "good" is defined.

To my wife and two daughters.

Acknowledgements

I would like to thank the members of my committee, my research advisor, Dr. Jeffery Weir and my reader, Capt Philip Jenkins PhD, for their guidance throughout this arduous journey.

Clarence O. Williams III

Table of Contents

Page

List of Figures

List of Tables

Table Page

Table Page

Table Page

META LEARNING RECOMMENDATION SYSTEM FOR CLASSIFICATION

I. Introduction

Operations Research (OR) has it origins in World War II, where throughout the war, upwards of 250 analysts were employed to solve complex problems like target assignment and bombing accuracy. The term OR itself owes its name to the British Royal Air Force, who used it to improve operations against German forces. The field's usage in the United States Department of War is a product of United States Army Air Forces Commanding General Henry "Hap" Arnold who championed the creation of the Operations Analysis Division of Air Staff Management Control Division on 31 December 1942. He saw the value in the integration of civilian experts and military officers in operational planning at the staff level [1].

Today, the field of OR has grown immensely with over 12,500 members of the Institute for Operations Research and the Management Sciences (INFORMS) society alone and with this growth, the scope of problems being investigated has exponentially grown in complexity due to revolutions in the storage and collection of information. A data driven approach is a new paradigm for handling analytical problems [2]. Meta Learning is considered using the features of a dataset to develop an overarching model about the features. The usage of meta learning for algorithm selection originates from Rice's model in which the purpose is to select a good or best algorithm for a particular problem [3].

One of the first usages of meta learning for regression problems was the METAL Project, where the purpose was used to select the most appropriate machine learning

algorithm for a given dataset using features extracted from the dataset [4]. Other current applications of meta modeling include multivariate time-series load forecasting, where meta learning is used to predict future electricity consumption and the identification of the appropriate load forecast model for building electricity consumption [5] [6].

Problem Statement

This research proposes an improvement on the meta-model recommendation system by adding classification problems to the candidate problem space with appropriate evaluation metrics for these additional problems. In its current implementation the meta learner has algorithms suited for continuous responses. Therefore, to add classification problems, classification algorithms will be added to the framework. The intent of this thesis is not to predict the absolute expected performance for algorithm recommendation but rather predict the relative performance among algorithms [7]. Additionally, the research seeks to answer the following questions:

1. Can the meta learner correctly make recommendations when classification and prediction are included as available algorithms? In order to assess this question, the algorithms suited for regression will have its output treated as class probabilities and the threshold will be set for class prediction.

2. Is normalized root mean square error (NRMSE) a suitable evaluation metric when using the meta learning recommendation system to rank algorithm selection for classification problems?

3. If NRMSE is insufficient, what are suitable evaluation metrics for the meta learning recommendation system to employ for ranking algorithm selection for classification problems?

II. Literature Review

Overview

This chapter reviews previously published literature on machine learning algorithms used for the meta recommendation framework and performance metrics. Relevant meta-modeling techniques will be discussed as well as an overview of meta features of interest for this study. The machine learning algorithms presented here are not an exhaustive list of all available algorithms but are only the techniques relevant to the framework.

Rice's Model

Rice proposed a formulation of abstract models to guide the selection of a best or good algorithm. This abstraction is shown in Figure 1 and it seeks to determine the selection mapping $S(f(x))$. In this model for algorithm selection, the four elements are the problem space P, feature space F, algorithm space A and performance space Y. For a meta learner, the problem space is the collection of all datasets used for training the learner. The feature space is all of the quantifiable properties. This model assumes that problems with the same features will have similar performance when applying algorithms. However, the selection of the best features to characterize a problem is a nebulous task. These features are essential to predict a best performing algorithm. For example, for solving a system of equations $Ax = b$, Rice states that an analyst can select a good algorithm to solve this system by examining the features of the system, such as sparsity, diagonally dominant, positive definite, condition number, etc. The algorithm space A,is all algorithms under consideration for the construction of the meta learner. Lastly, the performance space consists of all metrics used to evaluate the algorithms $a \in A$ against the problems $x \in P$. The model's usage of

meta learning is to frame the problem in order to give better results then randomly picking a algorithm [3].

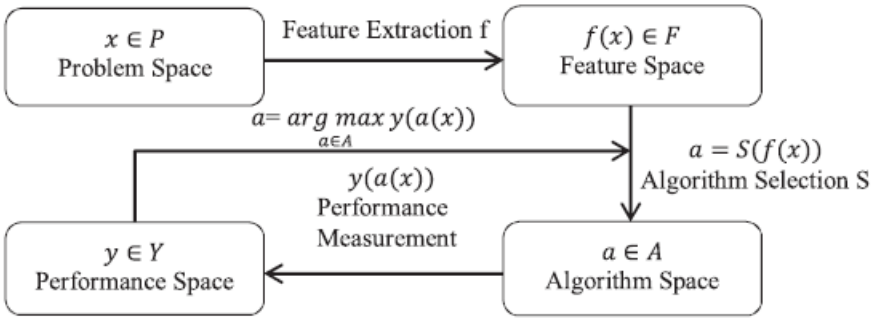

Figure 1. Rice's Model [3]

Meta Learning Framework

The Meta Learning Based Recommendation System was first proposed by Cui et al. and is shown in Figure 2 [8]. This new framework is a modification of Rice's model shown in Figure 1 and modifies the model by adding the feature reduction of the meta-features and the usage of members of the performance space to rank the algorithms in the algorithm space.

In the framework present in Figure 2, the model-based algorithm refers to the usage of an artificial neural network as the meta learner. While, the instance based algorithm refers to the usage of k nearest neighbors with $k \in \{1, 3\}$ as the meta learner.

Begin: Given new instance $x_{new} \in P$, meta-examples $x \in P$, feature reduction d, meta-learner algorithm R, accuracy performance measurement $y \in Y$

Step 1: Conduct feature extraction $f(x_{new})$

Step 2: Conduct feature reduction $d(f(x_{new}))$

Step 3: Meta-learning: find rankings $\{a_1, a_2, \dots, a_k\}$, where $a \in A$, k is the number of algorithm candidates, such that $y(a_{k-1}(x_{new})) \geq y(a_k(x_{new}))$

Case meta-learner R **OF**

Model-based algorithm:

$$a = R(d(f(x)), d(f(x_{new})), y(x))$$

Instance-based algorithm:

$$a = R(d(f(x)), d(f(x_{new})))$$

End Case

Step 4: Return the final rankings of recommendation: $\{a_1, a_2, \dots, a_k\}$

Figure 2. Meta Learning Based Recommendation System Framework [8]

Meta-Features

In order to properly select a model framework, a body of features are identified that can explain the underlying structure of the dataset. Meta features are classified as simple, statistical or information theoretic [9]. Some meta features of interest are:

- Number of discrete columns
- Minimum number of factors among discrete columns
- Maximum number of factors among discrete columns
- Average number of factors among discrete columns
- Number of continuous columns
- Gradient average

  For an N dimensional array A, the gradient is the derivatives of A with respect to each dimension. This measures the steepness of A in each dimension [8].

- Gradient maximum
- Gradient standard deviation
- Gradient minimum

Additional meta features could be the Mean of response values, [8]

$$\bar{f} = 1/N \sum_{i=1}^{N} f_i, \tag{1}$$

or the standard deviation of response values, [8]

$$SD(f) = \sqrt{1/(N-1) \sum_{i=1}^{N} (f_i - \bar{f})^2}, \tag{2}$$

which is the square root of variance which is the measure of the variability or amount of spread in the distribution of the response [10].

Machine Learning Algorithms

The machine learning algorithms used in this research are presented next. Any of the following algorithms can used for the construction of the meta learner.

Multiple Linear Regression.

Linear regression is used when a input vector is used to predict a response. They have the form

$$f(X) = \beta_0 + \sum_{j=1}^{p} X_j \beta_j, \tag{3}$$

where X_j is the input vector and β_j is the regressor coefficients. One method to estimate the regressor coefficients is to employ least squares, which finds the regressor

coefficients that minimize the residual sum of squares.

$$RSS(\beta) = \sum_{i=1}^{N}(y_i - f(x_i))^2. \tag{4}$$

Since $\mathbf{X}$ is a $N \times (p+1)$ matrix and y is a $(N \times 1)$ matrix, Equation (4) is rewritten as follows:

$$RSS(\beta) = (\mathbf{y} - \mathbf{X}\beta)^T(\mathbf{y} - \mathbf{X}\beta). \tag{5}$$

Differentiating Equation (5) with respect to β and setting the derivative equal zero yields,

$$\frac{\partial RSS}{\partial \beta} = -2\mathbf{X}^T(\mathbf{y} - \mathbf{X}\beta) = 0. \tag{6}$$

The solution to Equation (6) is, [11]

$$\hat{\beta} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}. \tag{7}$$

Regularized Regression.

Like linear regression, ridge regression is used when a input vector is used to predict a response. The key difference is that an additional term has been added to the objective function to penalize large regressor coefficients. The objective function for ridge regression is shown in Equation (8).

$$min \sum_{i=1}^{N}(y_i - f(x_i))^2 + \lambda \sum_{i=1}^{N} \beta_i^2. \tag{8}$$

In Equation (8), λ is known as the tuning parameter. Varying this parameter will change the regressor coefficients. Typically, λ is tuned using a grid search[11].

K Nearest Neighbors.

K Nearest Neighbors (KNN) classification is a supervised machine learning algorithm that was first used by Fix and Hodges in 1951 [12]. It is a lazy learner which means it is an instance based learning algorithm in which no model is fit [13].
The algorithm for KNN classification is as follows [13]:

1. Choose k and a distance metric. The most commonly used distance metric is the 2 norm, which is euclidean distance. This metric is defined in Equation (9) [11].

$$d_{(i)} = ||x_i - x_0||_2 = \sqrt{(x_i - x_0)^2}. \tag{9}$$

2. Find the k-nearest neighbors of the training example for classification

3. Assign class label

   The algorithm uses the conditional probability of an observation belonging to class j based on the fraction of training examples in the training set who belong to class j, that is,

$$P(Y = j|X = x_0) = \frac{1}{K} \sum_{i \in N} I(y_i = j). \tag{10}$$

   The algorithm then predicts the label of the observation by assigning it to the class that has the largest probability [14]. In Equation (10), the summation is used with indicator function to count observations that are have class j label.

The optimal value of k is problem dependent and has been explored in Hall's paper [15]. Per training observation, KNN classification requires Np operations, where N are the observations and p are the predictors to find the neighbors [11]. Therefore, KNN classification will be slow when there are ten of thousands of observations because each observation has a distance metric calculated.

Support Vector Machines.

Support Vector Machines (SVM) is a supervised machine learning technique that is used for classification. The algorithm creates the maximal separating hyperplane between two or more classes. Its current implementation to allow the classification of nonlinear separable data was created by Vladmir Vapnik and colleagues at AT&T Bell Labs [16]. In SVM, the objective is to find the hyperplane that creates the biggest margin between the training points for the classes. The margin is the distance between the separating hyperplane and the closest training examples for each class[13].
Let,

$$\mathbf{w}^T x = 1, \tag{11}$$

$$\mathbf{w}^T x = -1, \tag{12}$$

be the positive and negative hyperplane respectively. These hyperplanes can be rewritten using the equation for a plane as follows:

$$w_0 + \mathbf{w}^T x_{pos} = 1, \tag{13}$$

$$w_0 + \mathbf{w}^T x_{neg} = -1. \tag{14}$$

In Equations (13) and (14), x_{pos} and x_{neg} are training examples that fall behind the hyperplane that bears the name of the subscript. Subtracting Equation (14) from Equation (13) yields,

$$\mathbf{w}^T(x_{pos} - x_{neg}) = 2. \tag{15}$$

Normalizing Equation (15) by dividing it by the norm of $\mathbf{w}$ gives,

$$\frac{\mathbf{w}^T(x_{pos} - x_{neg})}{||\mathbf{w}||} = \frac{2}{||\mathbf{w}||}. \tag{16}$$

Equation (16) is the margin that will be maximized using nonlinear optimization.

Typically, the reciprocal of the right hand side of Equation (16) is minimized [13]. Therefore, the formulation to find the margin is written as,

$$min \quad \frac{1}{2}||\mathbf{w}||^2 \tag{17}$$

$$subject\ to \quad y^{(i)}(x_i^T\mathbf{w} + w_0) \geq 1 \quad \forall i. \tag{18}$$

Equation (18) means that observations that belong to the positive and negative classes should fall behind the corresponding hyperplane [13]. In 1995, Vapnik introduced ξ, which is a slack variable, to allow the relaxation of the linear constraints in equation 18. This new classification method is called soft-margin classification [16] [13].

Equations (19) and (20) give the non-linear program to find the margin for soft-margin classification. In Equation (20), ξ allows some points to be on the outside of the margin, if the classes overlap in the feature space. Additionally, ξ is the total proportional amount by which predictions fall on the outside of their margin. $\mathbf{w}$ is the support vector which is orthogonal to the hyperplane.

$$min \quad \frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{N}\xi_i \tag{19}$$

$$subject\ to \quad \xi \geq 0,\ yi(x_i^T\mathbf{w} + w_0) \geq 1 - \xi_i \forall i \tag{20}$$

Using Lagrange multipliers, the solution for $\mathbf{w}$ in the non-linear program presented in Equations (19) and (20) is

$$\hat{\mathbf{w}} = \sum_{i=1}^{n}\hat{\alpha}_i y_i x_i. \tag{21}$$

In Equation (21), $\hat{\alpha}_i$ is the lagrange multiplier, $0 \leq \hat{\alpha}_i \leq C$, and C is a cost param-

eter that influences the width of the boundary for classification. Larger values of C result in a smaller classification boundary. Regardless of their correct or incorrect classification, points near classification boundary are the support vectors [11].

Additionally, SVM uses a kernel function to increase the dimension of the features to create a linear boundary in a higher dimensional space [11]. A popular kernel used for this classifier is the radial basis kernel which is given in Equation (22).

$$k(x,\ y) = exp(-\gamma||x-y||^2), \quad where \ \ y > 0. \tag{22}$$

In Equation (22), x is the input data and y is the response. γ is a scaling parameter that influences the value of C [11].

Lastly, support vector classifiers have a time complexity of $O(m^2 \times n)$ to $O(m^3 \times n)$. Therefore, when the training data has hundreds of thousands of observations, the algorithm execution will be slow [17].

Support Vector Regression.

SVMs have also been adapted for regression by Drucker et al. in 1997 [18]. In this case the objective is to minimize the function

$$H(\beta, \beta_0) = \sum_{i=1}^{N} V(y_i - f(x_i)) + \frac{\lambda}{2}||\beta||^2. \tag{23}$$

The function V in Equation (23) is given by Equation (24). Its purpose is to only consider errors larger than ε which is analogous to the points being on the outside of the margin in the Support Vector Classifier [11].

$$V_\varepsilon(r) = \begin{cases} 0, & if \ |r| < \varepsilon \\ |r| - \varepsilon, & otherwise. \end{cases} \tag{24}$$

The regressor coefficients are

$$\hat{\beta} = \sum_{i=1}^{N} (\hat{\alpha}_i^* - \alpha_i) x_i, \tag{25}$$

and predictions $\hat{y}$ are given by

$$\hat{y} = \sum_{i=1}^{N} (\hat{\alpha}_i^* - \alpha_i) \langle x, x_i \rangle + \beta_0. \tag{26}$$

Clarke et al. has shown that SVR is an effective algorithm for meta modeling due to its ability to approximate the phenomenon under study by providing a prediction equation [19].

Naive Bayes Classifier.

The naive bayes classifier is a supervised machine learning algorithm for classification problems. For this algorithm, consider training examples $x_1, x_2, \ldots, x_n$ and class y that is binary. The probability of the training example belonging to class y can be found using Bayes Theorem which is shown in Equation [20].

$$P(y|(x_1, x_2, \ldots, x_n) = \frac{P((x_1, x_2, \ldots, x_n)|y)P(y)}{P((x_1, x_2, \ldots, x_n)}. \tag{27}$$

The class assignment uses the naive assumption, which is all features x_i are independent. Using this information,

$$P((x_1, x_2, \ldots, x_n)|y) = \prod_{i=1}^{n} P(x_i|y). \tag{28}$$

The predicted class $\hat{y}$ is simply the class that has the largest probability given the

input features.

$$\hat{y} = \underset{y}{\operatorname{argmax}} P(y) \prod_{i=1}^{n} P(x_i|y). \tag{29}$$

Principal Component Analysis.

Principal Component Analysis (PCA) was invented in 1901 by Karl Pearson [21]. It is currently employed as an unsupervised machined learning technique to reduce the dimensionality of the data, in order to decrease the execution time of other machine learning algorithms. In PCA, the unit eigenvectors, U, of the covariance matrix are used to project the data into the linear subspace spanned by the set of k vectors of U. The number of principal components is denoted by k. The objective function of PCA, given in Equation (30), is to minimize the reconstruction error [11].

$$Min \quad ||\sum_{i=1}^{n}(x_i - \mathbf{U}^T x_i)||^2. \tag{30}$$

In Equation (30), $\mathbf{U}^T$ is a projection matrix formed from the k eigenvectors of the covariance matrix. The steps for the PCA algorithm are as follows [13]:

1. Standardize the data

   Center feature columns to have zero mean with standard deviation one.

$$x_{std}^{(i)} = \frac{x^{(i)} - \mu_x}{\sigma_x}. \tag{31}$$

2. Compute the covariance matrix Σ

$$\Sigma = \frac{1}{m}\sum_{i=1}^{n}(x^{(i)})(x^{(i)})^T. \tag{32}$$

3. Obtain the eigenvalues and eigenvectors of the matrix Σ

   This is typically accomplished by using singular value decomposition (SVD). SVD will return the eigenvalues in descending order with associated eigenvectors in the same order [22]. Equation (33) shows the decomposition of a $m \times n$ matrix A using SVD.

$$A = USV^T. \tag{33}$$

   In Equation (33), U is an $m \times m$ orthogonal matrix and V is an $n \times n$ orthogonal matrix. The first r singular values of A are the diagonal entries of S. By definition an orthogonal matrix U is a matrix such that,

$$U^T U = I. \tag{34}$$

   Since U is invertible, the columns of U are linearly independent and form a basis [22]. The unit eigenvectors of the covariance matrix Σ are called the principal components.

4. Find the variance explained by each principal component.

   The variance explained of principal component j is,

$$\frac{\lambda_j}{\sum_{j=1}^{n} \lambda_j}, \tag{35}$$

   where λ_j is the eigenvalue of principal component j. Typically for dimension reduction chose the number of principal components so that the total variance explained by all of the components is at least 95% [17].

5. Let the k be the number of principal components chosen for change of basis. Project X into the linear subspace spanned by the set of k vectors of U by

choosing the number of principal components k.

Let the first k vectors of U be the change of coordinates matrix, $\mathscr{U}_{\mathscr{B}}$.

$$\mathbf{x} = \mathscr{U}[\mathbf{x}]_{\mathscr{B}}. \tag{36}$$

Left multiplication of $\mathbf{x} = \mathscr{U}[\mathbf{x}]_{\mathscr{B}}$ by $\mathscr{U}^{-1} = \mathscr{U}^T$ gives,

$$[\mathbf{x}]_{\mathscr{B}} = \mathscr{U}^T\mathbf{x}. \tag{37}$$

In Equation (37), $\mathscr{U}$ is the change of coordinate matrix and $[\mathbf{x}]_{\mathscr{B}}$ is the coordinate vector relative to $\mathscr{B}$ [22]. Let U_{reduce} be the matrix formed from the k vectors of U. The vectors of U_{reduce} are called the principal component loading vectors. The projection of $\mathbf{x}$, in this new space is,

$$\mathbf{z} = U_{reduce}^T\mathbf{x}. \tag{38}$$

The entries in the columns of $\mathbf{z}$ are called the principal component scores.

6. Project $\mathbf{z}$ back into original space to approximate $\mathbf{x}$ if all principal components were not used.

$$\mathbf{x}_{approx} = U_r\mathbf{z}. \tag{39}$$

Equation (39), is the key step employed before implementing any other machine learning algorithms. It is the dimension reduction.

Next the evaluation metrics for each problem in the problem space is presented. Cui et al. has shown that NRMSE is a suitable performance metric for datasets with a continuous response [8]. Therefore, that metric is included here as well. Additionally, the proposed performance metrics for the classification datasets are defined.

Evaluation Metrics

the model effectiveness for each algorithm $a \in A$ is evaluated using the performance metrics presented in this section. The meta learner employed by Cui et al. used Normalized Root Mean Square Error which is given in Equation (40) as a performance metric. Therefore, that metric will be used in this research to see if it is a suitable performance metric for classification problems.

Normalized Root Mean Square Error

$$\sqrt{\frac{\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}{N}}/(y_{max} - y_{min}). \tag{40}$$

Classifier Evaluation Metrics

There exists myriad potential evaluation metrics for the recommendation system for datasets which have binary output. A subset of these metrics are given in following sections.

Confusion Matrix.

The confusion matrix shows the classification of all training examples. In Table 1, $C_{0,0}$ is the number of true negatives, $C_{0,1}$ is the number of false positives, $C_{1,0}$ is the number of false negatives and $C_{1,1}$ is the number of true positives [23]. Additionally, the total classifier accuracy is given in the confusion matrix by dividing the sum of the entries in the main diagonal by the sum of each entry in the matrix.

Table 1. Confusion Matrix

	Predicted	
Truth	Class 1	Class 2
Class 1	$C_{0,0}$	$C_{0,1}$
Class 2	$C_{1,0}$	$C_{1,1}$

Precision.

Let $TP,\ FP,\ FN$ be true positive, false positive and false negative rate respectively. Precision, which is the accuracy of positive predictions, is defined in Equation (42) [17].

$$precision = \frac{TP}{TP + FP}. \tag{41}$$

Using the entries of Table 1, precision is,

$$precision = \frac{C_{1,1}}{C_{1,1} + C_{0,1}}. \tag{42}$$

Recall.

Recall is the ratio of positive instances that are correctly detected by the classifier. It is also called true positive rate (TPR) or sensitivity and is defined in Equation (43) [17].

$$recall = \frac{TP}{TP + FN}. \tag{43}$$

Again using the entries of Table 1, recall is,

$$recall = \frac{C_{1,1}}{C_{1,1} + C_{1,0}}. \tag{44}$$

A classifier with high recall but low precision will have many predicted labels that are incorrect when compared to the training labels. This classifier predicts many positives instances. On the other hand, a classifier with high precision but low recall will have many correct predictions when compared to the training labels but the classifier is predicting many negative instances [24]. Note, there is a precision recall trade off. Increasing recall will reduce precision and vice versa [17].

F1 Score.

Another metric to evaluate classifiers is the F1 score. It is a single metric that is useful if one value is desired to compare two classifiers. The F1 score is the harmonic mean of precision and recall and is defined in Equation (45) [17].

$$F1 = \frac{2}{\frac{1}{precision} + \frac{1}{recall}} = \frac{TP}{TP + \frac{FN+FP}{2}}. \tag{45}$$

In order to have a high F1 score the precision and recall must be high [17]. From Equation (45), it is apparent that the metric is bounded on the interval (0, 1]. However, by definition if precision and recall are undefined, F1 score is considered 0.

III. Methodology

Overview

This chapter describes the datasets used for this research and the steps to implement the meta learning framework. In this research, each dataset presented forms the candidate problem space. The machine learning algorithms K- Nearest Neighbors, Support Vector Machines, Naive Bayes Classifier, Linear Regression, Ridge Regression and Support Vector Regression are the algorithms implemented within this meta learning framework. Additionally, the evaluation criteria for the ranking of each algorithm for every dataset is provided.

Datasets

In order to assess the meta learner's ability to perform recommendation on classification problems, six datasets with a discrete response form the problem space.. These six datasets have a binary output and may have continuous and/or discrete features. The names of the datasets are provided in Table 2.

Table 2. Dataset Descriptions

Dataset	Name	Response
1	Heart	target
2	Spam	yesno_bin
3	Bank Personal Loan	Personal Loan
4	Framingham	TenYearCHD
5	Math Placement	CourseSuccess
6	Credit Card Fraud	isFraud

Dataset 1, Heart, originates from Kaggle and is a complete dataset. It has 303 rows and 14 columns including the response. The goal of this dataset is to predict the presence of heart disease in a patient [25]. Dataset 2, Spam, comes from Python's

pydatasets, which is a python implementation of R datasets found in the R Project for Statistical Computing software [26]. In the online documentation, the dataset is named spam7. This dataset is also complete and contains 460 rows and 7 columns including the response. This dataset's purpose is to predict if an email is spam. Dataset 3, Bank Personal Loan, originates from Kaggle and is again a complete dataset [27]. It has 5000 rows and 14 columns including the response. The goal of this dataset is to predict if a customer will accept a personal loan. Dataset 4, Framingham, originates from Kaggle [28]. This dataset contains missing records and in its original form, it has 4238 rows and 16 columns including the response. The goal of this dataset is to predict a person's ten year risk of future coronary heart disease where the prediction is binary.

Dataset 5, Math Placement, originates from pydatasets like Dataset 2 [29]. This dataset is missing records and in its original form, it has 2696 observations and 16 variables including the response. To produce dataset 5, the columns UID student is dropped because it is a unique identifier. The columns Gender, PSTAM and STAM are also dropped due to missing 2116, 1560, and 1460 records respectively. Next, the column grade is dropped because it is the letter grade associated with the response. Finally, the column Recommends is dropped because the information contained within this feature is redundant due to the presence of other features related to recommend. The goal of this dataset is the classification of Course Success.

Lastly, dataset 6, Credit Card Fraud, is available on Kaggle [30]. To construct Dataset 6, the following steps are completed. Two data sets, Transaction and Identity are provided to classify transactions as fraudulent. First, the Identity dataset and the Transaction dataset are joined on the unique identifier, TranactionID. Next, features that are not 60% filled are subsequently dropped and then rows that are incomplete are removed. Lastly, the joined dataset from step 2 contained 17 categorical

features which are dropped along with the unique identifier column. The response of this dataset is the column isFraud which is used to label a credit card transaction as fraudulent. The goal of this dataset is to predict if credit card transaction is fraudulent.

Meta Learning Framework

The Meta Learning Recommendation framework was first proposed by Cui et al. and is further refined in the AFIT Master's Thesis of Megan Woods [31]. The new framework in Figure 3 is a modification of Cui's framework shown in Figure 2. The current framework, is similar to Cui's framework in regards to extracting meta features f of members of the problem space P. However, it modifies the existing framework by implementing data cleaning and filtering of candidate problems C. These $c \in C$ may not meet the criteria to enable successful algorithm recommendation using the framework and are filtered out before implementation.

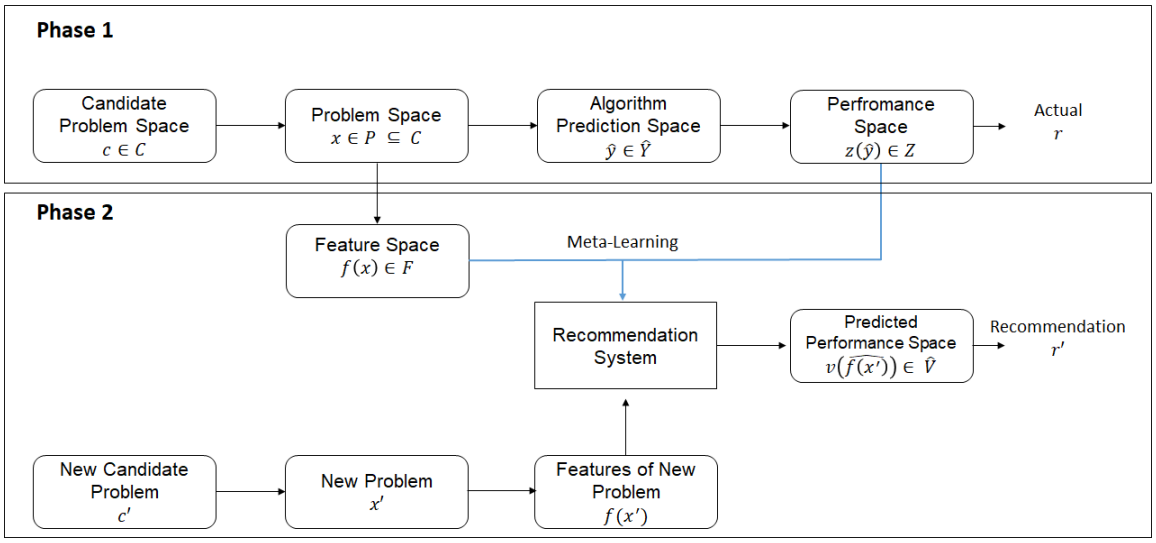

Figure 3. Meta Learning Framework [31]

The meta learning recommendation system for classification problems consists of two phases. The steps of phase one are as follows:

1. Candidate Problem Space

   The Candidate Problem space C is all problems suitable for classification. Since this set is large, it is subsetted to form the problem space P which contains the problems under study for this thesis.

2. Algorithm Prediction Space

   The machine learning algorithms K Nearest Neighbors, Support Vector Machines, Naive Bayes Classifier, Linear Regression, Ridge Regression and Support Vector Regression form the algorithm space A. The six algorithms are subsequently applied to each member of the problem space with normalized root mean square error (NRMSE), F1 score, precision and recall being the performance metrics captured for each dataset. NRMSE is calculated by comparing the output of the respective algorithm to the class labels. Since Linear Regression, Ridge Regression and Support Vector Regression are not naively suited for classification, the output of these three algorithms are treated as class probabilities and the threshold for class prediction is set to 0.5.

   Since the default parameters of each algorithm may change over time, the specific parameters used to for algorithm are as follows:

   - Support Vector Machine:
     sklearn.svm.SVC(kernel='rbf', degree=3, gamma='scale', coef0=0.0, tol=0.001, C=1.0, epsilon=0.1, shrinking=True, cache_size=200, verbose=False, max_iter=-1) Scale is given by Equation 46.

$$Scale = \frac{1}{(number\ of\ features * X.var())}. \tag{46}$$

- K- Nearest Neighbors

  sklearn.neighbors.KNeighborsRegressor(n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=30, p=2, metric='minkowski', metric_params=None, n_jobs=None

- Naive Bayes Classifier sklearn.naive_bayes

  .GaussianNB(priors=None, var_smoothing=1e-09)

- Support Vector Regression

  The settings are the same as as SVM.

- Ridge Regression

  sklearn.linear_model.Ridge(alpha=1.0, fit_intercept=True, normalize=False, copy_X=True, max_iter=None, tol=0.001, solver='auto', random_state=None)

- Linear Regression

  sklearn.linear_model.LinearRegression(fit_intercept=True, normalize=False, copy_X=True, n_jobs=None)

3. Recommendation

   Each algorithm has its performance ranked when applied to each of the six datasets in the problem space. This ranking is repeated for each performance metric to give a separate ranking for each metric. When using NRMSE, the best algorithm is the one with the lowest value. This algorithm r is given by Equation (47).

$$r = \underset{a \in A}{\operatorname{argmin}}(z(a(x))). \tag{47}$$

   When F1 score, precision or recall is the performance metric, the best algorithm is the one with the largest value. In this case, the best algorithm r is given by

Equation (48).

$$r = \underset{a \in A}{\operatorname{argmax}}(z(a(x))). \tag{48}$$

In phase 2 of the meta learning recommendation system, the following steps occur

1.1 Meta Feature Extraction

Each of the members of the problem space have information extracted to provide information about it's structure. The following meta features are extracted:

- Number of Rows
- Number of Columns
- Rows to Columns Ratio
- Number of Discrete Columns
- Maximum number of factors among discrete columns
- Minimum number of factors among discrete columns
- Average number of factors among discrete columns
- Number of continuous columns
- Gradient average
- Gradient minimum
- Gradient maximum
- Gradient standard deviation

1.2 Dimension Reduction

The feature space is reduced using principal component analysis to remove multicollinearity.

2. Meta Learning

   A new dataset is formed where each row is the collection of the 12 meta features extracted from one dataset of the six datasets. These twelve features together form the feature space F. Since there are six datasets in the problem space, this new dataset has dimensionality 6×12.

3. Recommendation System Construction

   Support Vector Regression (SVR) is the meta learner that trains the recommendation system using the meta features as inputs with a metric of the performance space for each algorithm as output. Leave one out (LOO) validation gives the final performance metric prediction for each algorithm. In this instance of LOO, five out of the six datasets trains the recommendation system and one dataset is withheld for the test set. For example, for dataset 1, the meta features extracted from datasets 2 through 6 are the training datasets to build the recommendation system. SVR fits a model for the six algorithms in the algorithm space using NRMSE as the response. This process is then repeated using F1 score and any other member of the performance space as the response when training the linear regression meta modeler.

4. Performance Prediction and Recommendation

   The recommendation system predicts the performance of the machine learning algorithms K Nearest Neighbors, Support Vector Machines, Naive Bayes Classifier, Linear Regression, Ridge Regression and Support Vector Regression for each member of the problem space. Each algorithm, $a \in A$, has its performance ranked when applied to each of the six datasets in the problem space. This ranking is repeated for each performance metric to give separate rankings. Similar to phase one, when using NRMSE, the best algorithm is the one with

the lowest value, that is the recommendation r' is given by Equation (49),

$$r' = \underset{a \in A}{\operatorname{argmin}}(\widehat{v(f(x'))}). \tag{49}$$

When F1 Score is the performance metric, the best algorithm is the one with the largest value, that is the recommendation r' is given by Equation (50).

$$r' = \underset{a \in A}{\operatorname{argmax}}(\widehat{v(f(x'))}). \tag{50}$$

Furthermore, NRMSE is calculated in two different manners. The first uses the class predictions, $\hat{y} \in \{0, 1\}$ for the calculation of the metric. In the second, the class probabilities, $\hat{y} \in [0, 1]$, returned by each $a \in A$, are used to calculate the metric. In either case, the difference between y_{max} and y_{min} is always one and this metric is equivalent to root mean square error.

Evaluation

The meta learner recommendation systems final evaluation is the hit ratio when using each predicted metric. For a given dataset, the hit ratio is the number of matches of the best performing algorithm with recommended best algorithm. Additionally, the hit ratio is relaxed to consider a hit if the recommended best algorithm's actual performance metric is within 0.01%, 0.05% and 0.10% of the actual best algorithm.

IV. Analysis and Results

Overview

In this chapter, each dataset in the candidate problem space forms the problem space P. The machine learning algorithms K-Nearest Neighbors, Support Vector Machines, Naive Bayes Classifier, Linear Regression, Ridge Regression and Support Vector Regression forms the algorithm space A. Each algorithm $a \in A$ is applied to all candidate problems in the problem space and performance is evaluated using the performance measurements, normalized root mean square error, F1 score, precision and recall.

Meta Features

The following features are extracted from each candidate problem to construct the meta learning recommendation system.

- Number of Rows
- Number of Columns
- Rows to Columns Ratio
- Number of Discrete Columns
- Maximum number of factors among discrete columns
- Minimum number of factors among discrete columns
- Average number of factors among discrete columns
- Number of continuous columns

- Gradient average
- Gradient minimum
- Gradient maximum
- Gradient standard deviation

Table 8 gives the time in seconds to execute each algorithm $a \in A$ for each problem $p \in P$.

Table 3. Algorithm Execution Time in Seconds

Dataset	SVM	KNN	Naive Bayes	SVR	Ridge Regression	Linear Regression
Heart	0.0060	0.0080	0.0050	0.0060	0.0070	0.0060
Spam	0.1899	0.0790	0.0080	0.2019	0.0120	0.0130
Bank Personal Loan	0.1139	0.1459	0.0080	0.1879	0.0180	0.0130
Framingham	0.2588	0.1799	0.0100	0.2828	0.0130	0.0150
Math Placement	0.0630	0.0500	0.0080	0.0770	0.0100	0.0100
Credit Card Fraud	746.88	508.85	0.5577	639.77	0.2858	1.0366

Table 4 shows the meta features extracted from each problem $p \in P$. These meta features are scaled using Python standard scaler before training the meta modeler.

Table 4. Meta Features

Data	Rows	Columns	Rows-Cols Ratio	Number Discrete	Max Num Factors	Min Num Factors	Avg Num Factors	Number Continuous	Gradient Avg	Gradient Min	Gradient Max	Gradient Std
Heart	303	13	23.3077	11	49	2	14.0909	2	-6.0079	-282	281	64.9579
Spam	4601	6	766.8333	6	964	142	472.6667	0	-70.7983	-15841	13.739	297.0871
Bank Personal Loan	5000	12	416.6667	12	467	2	99.25	0	-4.8177	-48325.4	48325	19017.3793
Framingham	3656	15	243.7333	14	241	2	65.1429	1	7.2688	-253	319	46.5080
Math Placement	1788	9	198.6667	8	237	2	44.75	1	-1.7399	-431.5	432	88.8450
Credit Card Fraud	75988	229	331.8253	226	8694	1	841.5177	3	13110.43	-7903595	15666682	573053.3887

Table 5. Scaled Meta Features

Data	Rows	Columns	Rows-Cols Ratio	Number Discrete	Max Num Factors	Min Num Factors	Avg Num Factors	Number Continuous	Gradient Avg	Gradient Min	Gradient Max	Gradient Std
Heart	-0.5015	-0.4321	-0.7971	-0.4296	-0.5171	-0.4434	-0.7120	-0.1768	-0.4384	0.4517	-0.4488	-0.4655
Spam	-0.3463	-0.5186	1.9924	-0.4916	-0.2269	2.2360	0.7432	-1.2374	-0.4516	0.4464	-0.4489	-0.4644
Bank Personal Loan	-0.4948	-0.4445	-0.7316	-0.4420	-0.4980	-0.4434	-0.6609	-0.1768	-0.4382	0.4354	-0.4407	-0.3762
Framingham	-0.3804	-0.4074	0.0299	-0.3924	-0.4562	-0.4434	-0.5500	-0.7071	-0.4357	0.4517	-0.4488	-0.4655
Math Placement	-0.5086	-0.4321	-0.8539	-0.4792	-0.5263	-0.4434	-0.7341	1.9445	-0.4720	0.4509	-0.4488	-0.4634
Credit Card Fraud	2.2317	2.2347	0.3604	2.2349	2.2244	-0.4625	1.9138	0.3536	2.2359	-2.2360	2.2361	2.2349

Due to potential redundant information being contained within the meta features and the number of features being greater than the number of datasets, the dimensionality of the meta features is reduced using PCA. Figure 4 graphs the variance explained by each principal component. Four principal components explain 100% of the variability in the meta features. However, three principal components are chosen for the dimension reduction to graph the meta features in 3-dimensional space. These three principal components explain 99.7% of the variability in the data. The loading vectors for these principal components are shown in Table 6.

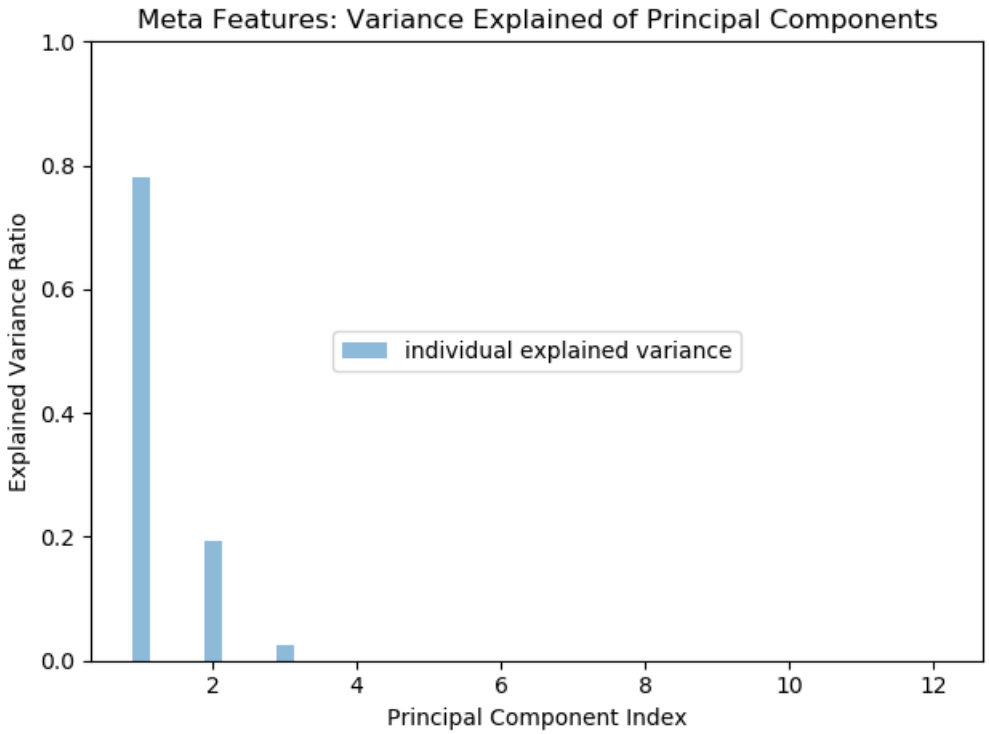

Figure 4. Principal Component Analysis of Meta Features

Table 6. Principal Component Loading Vector

Principal Component Loading Vector	Rows	Columns	Rows-Cols Ratio	Number Discrete	Max num factors	Min num factors	Avg num factors	Number Continuous	Gradient Avg	Gradient Min	Gradient Max	Gradient Std
1	0.3257	0.3265	-0.0030	0.3265	0.3249	-0.0677	0.2807	0.2555	0.3266	-0.3266	0.3266	0.3263
2	0.0332	-0.0118	0.6458	-0.0076	0.0662	0.5997	0.3314	-0.3281	0.0026	-0.0071	0.0059	0.0069
3	0.1126	0.0555	0.2998	0.0650	0.0556	-0.6280	-0.1408	-0.6765	0.0459	-0.0503	0.0470	0.0915

The projection of the meta features into the principal components space is shown in Figure 5. Additionally, Table 7 gives the coordinates for each dataset shown in Figure 5.

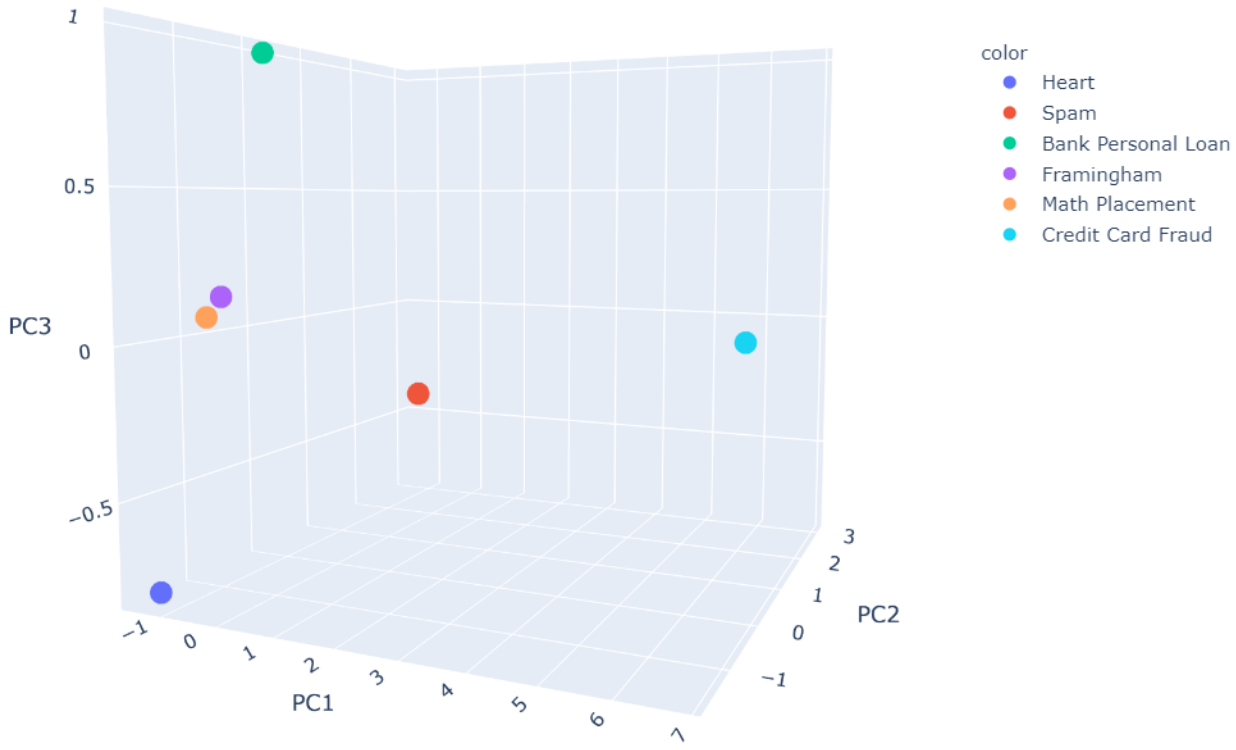

Figure 5. Meta Features Projected in Principal Component Space

Table 7. Meta Features in Principal Component Space

Dataset	PC1	PC2	PC3
Heart	-1.2211	-1.7051	-0.7891
Spam	-1.3708	3.1317	-0.4216
Bank Personal Loan	-1.4891	0.1232	0.9887
Framingham	-1.3349	-0.7152	0.1298
Math Placement	-1.4249	-0.8649	0.0637
Credit Card Fraud	6.8408	0.0303	0.0285

All datasets except Credit Card Fraud are nearly coplanar in the Principal Component 1 plane. The euclidean distance between the Framingham and Math Placement

datasets is 0.1868. Therefore, the algorithms are expected to perform similarly on these two datasets if the distribution of the response is similar between the two.

Table 8 gives the time to execute each $a \in A$ for each $p \in P$. Since KNN, SVM and SVR scale poorly with the size of the dataset, Credit Card fraud dataset has the worst time performance.

Table 8. Algorithm Execution Time in Seconds

Dataset	SVM	KNN	Naive Bayes	SVR	Ridge Regression	Linear Regression
Heart	0.0060	0.0080	0.0050	0.0060	0.0070	0.0060
Spam	0.1899	0.0790	0.0080	0.2019	0.0120	0.0130
Bank Personal Loan	0.1139	0.1459	0.0080	0.1879	0.0180	0.0130
Framingham	0.2588	0.1799	0.0100	0.2828	0.0130	0.0150
Math Placement	0.0630	0.0500	0.0080	0.0770	0.0100	0.0100
Credit Card Fraud	746.88	508.85	0.5577	639.77	0.2858	1.0366

NRMSE

In this section, the six datasets in the problem space P, have their algorithm performance predicted using the metric NMRSE. The true algorithm performance is ranked for each dataset using this metric and the algorithm ranking returned by the meta learner recommendation system are evaluated.

Table 9 shows each algorithms' performance using the metric NRMSE. Note that this metric is not normally to evaluate classifiers. The rankings of each algorithm $a \in A$ is given in Table 10. Using NRMSE, SVR is the true best performing algorithm in 50% of the datasets, while its classification counterpart SVM is the worst performing algorithm in 33.33% of the datasets. Overall, SVR is always in the top third performing algorithms, while SVM is in the bottom two, in 66.7% of the datasets.

Table 9. Dataset Actual NRMSE

Dataset	SVM	KNN	Naive Bayes	SVR	Ridge Regression	Linear Regression
Heart	0.4149	0.4527	0.3841	0.4049	0.4049	0.4049
Spam	0.3787	0.3670	0.4950	0.3801	0.4736	0.4736
Bank Personal Loan	0.1987	0.2145	0.3178	0.1857	0.2500	0.2500
Framingham	0.5638	0.4076	0.4167	0.3904	0.3904	0.3904
Math Placement	0.5493	0.5364	0.5403	0.4930	0.5138	0.5138
Credit Card Fraud	0.2740	0.2196	0.2979	0.2088	0.2454	0.2459

Table 10. Dataset Actual NRMSE Ranking

Dataset	SVM	KNN	Naive Bayes	SVR	Ridge Regression	Linear Regression
Heart	5	6	1	3	3	3
Spam	2	1	6	3	4.5	4.5
Bank Personal Loan	2	3	6	1	4.5	4.5
Framingham	6	4	5	2	2	2
Math Placement	6	4	5	1	2.5	2.5
Credit Card Fraud	5	2	6	1	3	4

To predict the performance metric NRMSE, SVR is the meta learner with the meta features as the input and each algorithms' NRMSE as the target variable. Leave one out validation gives the final predicted NRMSE of each dataset.

The final predicted NRMSE of the meta models is given in Table 11 and the predicted algorithm ranking is given in Table 12. The meta learner ranks SVR as the top performing algorithm for 66.7% of the datasets, which could be due to the true performance of that algorithm ranking in the top three for all datasets.

Table 11. Dataset Predicted NRMSE

Dataset	SVM	KNN	Naive Bayes	SVR	Ridge Regression	Linear Regression
Heart	0.5857	0.5044	0.4508	0.4552	0.4520	0.4520
Spam	0.2010	0.2472	0.4026	0.2290	0.3138	0.3139
Bank Personal Loan	0.4593	0.4290	0.4376	0.3877	0.4095	0.4095
Framingham	0.4328	0.4237	0.4377	0.3815	0.4071	0.4071
Math Placement	0.4471	0.3353	0.3964	0.2959	0.3487	0.3487
Credit Card Fraud	0.3815	0.3755	0.4290	0.3394	0.3819	0.3819

Table 12. Dataset Predicted NRMSE Rankings

Dataset	SVM	KNN	Naive Bayes	SVR	Ridge Regression	Linear Regression
Heart	6	5	1	4	2	3
Spam	1	3	6	2	4	5
Bank Personal Loan	6	4	5	1	2	3
Framingham	5	4	6	1	3	2
Math Placement	6	2	5	1	3	4
Credit Card Fraud	3	2	6	1	4.5	4.5

The evaluation of the meta learner using NRMSE as a performance metric for classification problems is given by Table 13. Using NRMSE as the performance metric with classification problems, causes the system to recommend the usage of the actual best performing algorithm in two out of the six datasets. A priori knowledge of each $a \in A$ performance for all $p \in P$, allows the NRMSE of the recommendation to be compared with the known best algorithm. In Table 13, relaxing the tolerance of a hit to be within 10% of the actual best NRMSE, improves the hit rate to 50%.

Table 13. NRMSE Recommendation Rating

					Epsilon		
Model	Actual Best NRMSE	Best Algorithm	Recommended Algorithm	NRMSE of Recommendation	0.01	0.05	0.1
Heart	0.3841	NB	NB	0.3841			
Spam	0.3670	KNN	SVM	0.3787			
Bank Personal Loan	0.1857	SVR	SVR	0.1857			
Framingham	0.3904	SVR	SVR	0.3904			
Math Placement	0.4930	SVR	SVR	0.4930			
Credit Card Fraud	0.2088	SVR	SVR	0.2088			

NRMSE using Class Probabilities

In this section, the six datasets in the problem space P, have their algorithm performance predicted using the metric NMRSE calculated using class probabilities. In this case, the predicted values $\hat{y} \in [0, 1]$. The true algorithm performance is ranked for each dataset using this metric and the algorithm ranking returned by the meta learner recommendation system are evaluated.

Table **??** shows each algorithms performance using the metric different calculation of NRMSE. Like the previous section, note that this metric is not normally to evaluate classifiers. The rankings of each algorithm $a \in A$ is given in Table **??**.

Table 14. Dataset Actual NRMSE using Class Probabilities

Dataset	SVM	KNN	Naive Bayes	SVR	Ridge Regression	Linear Regression
Heart	0.3436	0.3781	0.3514	0.3456	0.3390	0.3390
Spam	0.3303	0.3403	0.4809	0.3380	0.4020	0.4020
Bank Personal Loan	0.1593	0.1909	0.2901	0.1573	0.2232	0.2232
Framingham	0.3484	0.3691	0.3946	0.3527	0.3452	0.3452
Math Placement	0.4119	0.4473	0.4762	0.4259	0.4232	0.4232
Credit Card Fraud	0.2083	0.2087	0.2973	0.2030	0.2167	0.2165

Table 15. Dataset Actual NRMSE using Class Probabilities Ranking

Dataset	SVM	KNN	Naive Bayes	SVR	Ridge Regression	Linear Regression
Heart	3	6	5	4	1	2
Spam	1	3	6	2	5	4
Bank Personal Loan	2	3	6	1	5	4
Framingham	3	5	6	4	1	2
Math Placement	1	5	6	4	2	3
Credit Card Fraud	2	3	6	1	5	4

To predict the performance metric, NRMSE calculated using the class probabilities, SVR is the meta learner with the meta features as the input and each algorithms' NRMSE as the target variable. Leave one out validation gives the final predicted NRMSE of each dataset.

The final predicted NRMSE using class probabilities of the meta models is given in Table **??** and the predicted algorithm ranking is given in Table **??**. SVM ranks

in the top three for all datasets and Naive Bayes ranked last in % datasets. Due to SVM's performance on the datasets, the recommendation system predicted it as top performing algorithm in 66.67% cases.

Table 16. Dataset Predicted NRMSE using Class Probabilities

Dataset	SVM	KNN	Naive Bayes	SVR	Ridge Regression	Linear Regression
Heart	0.3436	0.3794	0.3855	0.3674	0.3229	0.3229
Spam	0.2335	0.2625	0.3832	0.2236	0.3217	0.3216
Bank Personal Loan	0.3116	0.3448	0.3891	0.3245	0.3228	0.3228
Framingham	0.3063	0.3410	0.3855	0.3187	0.3232	0.3232
Math Placement	0.2538	0.2845	0.3855	0.2550	0.3094	0.3093
Credit Card Fraud	0.2856	0.3191	0.3855	0.2916	0.3232	0.3232

Table 17. Dataset Predicted NRMSE using Class Probabilities Ranking

Dataset	SVM	KNN	Naive Bayes	SVR	Ridge Regression	Linear Regression
Heart	3	5	6	4	2	1
Spam	2	3	6	1	5	4
Bank Personal Loan	1	5	6	4	3	2
Framingham	1	5	6	2	3	4
Math Placement	1	3	6	2	5	4
Credit Card Fraud	1	3	6	2	5	4

The evaluation of the meta learner using this different NRMSE as a performance metric for classification problems is given by Table **??**. Using NRMSE as the performance metric with classification problems causes the system to recommend the usage of the actual best performing algorithm in only one out of the six datasets. However, despite this low hit ratio, the actual NRMSE of the recommendation is within 5% of the true best. Thus, once the hit ratio is relaxed to the recommendation being 5% of the true best, then it is improved to 100%.

Table 18. NRMSE using Class Probabilities Recommendation Rating

					Epsilon		
Model	Actual Best NRMSE	Best Algorithm	Recommended Algorithm	NRMSE of Recommendation	0.01	0.05	0.1
Heart	0.3390	RR	LR	0.3390			
Spam	0.3303	SVM	SVR	0.3380			
Bank Personal Loan	0.1573	SVR	SVR	0.1593			
Framingham	0.3452	RR	SVM	0.3484			
Math Placement	0.4119	SVM	SVM	0.4119			
Credit Card Fraud	0.2030	SVR	SVM	0.2083			

F1 Score

In this section, the six datasets in the problem space P, have their algorithm performance predicted using the metric F1 score. F1 score is a single metric that is useful if one value is desired to compare two classifiers. It is the harmonic mean of precision and recall.

Since the algorithms support vector regression, linear regression and ridge regression are not normally used for classification, a threshold is set to assign class labels. Figure 14 shows that a decision threshold of 0.5 is not optimal for the Credit Card Dataset using SVR. Additional, graphs of the F1 score versus the decision threshold for the other dataset algorithm combinations are included in Appendix A.

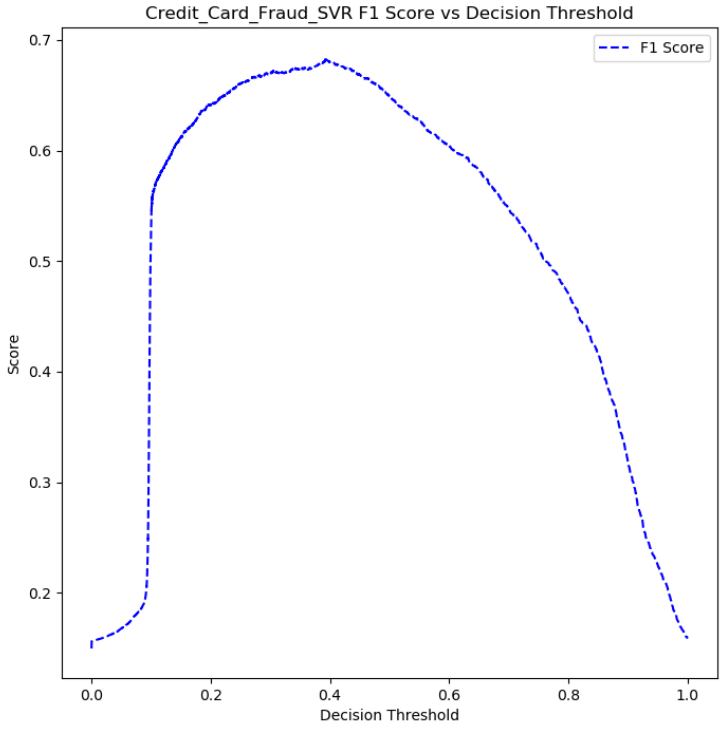

Figure 6. SVR Credit Card Fraud F1 vs Threshold

Table 19 shows each algorithms performance using the metric F1 score. The rankings of each algorithm $a \in A$ is given in Table 20. SVM and KNN are typically the top performing algorithms except for the Heart dataset.

Table 19. Dataset Actual F1 Score

Dataset	SVM	KNN	Naive Bayes	SVR	Ridge Regression	Linear Regression
Heart	0.8346	0.8175	0.8636	0.8485	0.8551	0.8551
Spam	0.8101	0.8166	0.5933	0.7966	0.6158	0.6158
Bank Personal Loan	0.8159	0.6993	0.5511	0.7890	0.5247	0.5247
Framingham	0.3708	0.1413	0.2743	0.0823	0.0089	0.0089
Math Placement	0.7494	0.7972	0.7649	0.8362	0.8219	0.8219
Credit Card Fraud	0.5994	0.6206	0.4970	0.6490	0.4298	0.4292

Table 20. Actual F1 Score Rankings

Dataset	SVM	KNN	Naive Bayes	SVR	Ridge Regression	Linear Regression
Heart	5	6	1	4	2.5	2.5
Spam	2	1	6	3	4.5	4.5
Bank Personal Loan	1	3	4	2	5.5	5.5
Framingham	1	3	2	4	5.5	5.5
Math Placement	6	4	5	1	2.5	2.5
Credit Card Fraud	3	2	4	1	5	6

Like the previous performance metrics, to predict F1 score, SVR with a radial basis function kernel is the meta learner with the meta features projected to 3-dimension space as the input and each algorithms' F1 score as the target variable. Additionally, the default parameters of python's implementation of SVR are used and the hyperparameter gamma is set to scale which is given by Equation (46) and the penalty hyperparameter C is set to one. Leave one out validation gives the final predicted F1 score of each dataset. The final predicted F1 score of the meta models is given in Table 21 and algorithm ranking is given in Table 22. Similar to the usage of NRMSE as performance metric, the recommendation gives SVR as the best or second best performing algorithm for all six datasets because its true performance is in the top three in 66.67% of the datasets used to construct the meta learner.

Table 21. Dataset Predicted F1 Score

Dataset	SVM	KNN	Naive Bayes	SVR	Ridge Regression	Linear Regression
Heart	0.6143	0.7181	0.5824	0.7520	0.5609	0.5607
Spam	0.7716	0.6392	0.5403	0.6741	0.5699	0.5698
Bank Personal Loan	0.5899	0.6789	0.5690	0.7045	0.6653	0.6653
Framingham	0.7324	0.7191	0.7084	0.7487	0.7119	0.7119
Math Placement	0.6952	0.6257	0.5857	0.6877	0.5704	0.5704
Credit Card Fraud	0.7323	0.6984	0.6703	0.7371	0.6894	0.6894

Table 22. Predicted F1 Score Rankings

Dataset	SVM	KNN	Naive Bayes	SVR	Ridge Regression	Linear Regression
Heart	3	2	4	1	5	6
Spam	1	3	6	2	4	5
Bank Personal Loan	5	2	6	1	3	4
Framingham	2	3	6	1	5	4
Math Placement	1	3	4	2	6	5
Credit Card Fraud	2	3	6	1	4.5	4.5

The evaluation of the meta learning recommendation system using F1 score as a performance metric is given by Table 23. Using F1 score as the performance metric with classification problems results in a worse hit ratio than using NRMSE. The recommendation system correctly selects the true best performing algorithm in one case. If the criteria for a hit is relaxed to the system's recommendation being within 5% of the actual best algorithm, then the hit ratio improves to 66.67%. Additionally, relaxing the tolerance of a hit to being within 5% of the actual best algorithm, improves the hit rate to 66.67%.

Table 23. F1 Score Recommendation Rating

					Epsilon		
Model	Actual Best Precision	Best Algorithm	Recommended Algorithm	Precision of Recommendation	0.01	0.05	0.1
Heart	0.8636	NB	SVR	0.8485			
Spam	0.8166	KNN	SVM	0.8101			
Bank Personal Loan	0.8159	SVM	SVR	0.7890			
Framingham	0.3708	SVM	SVR	0.0823			
Math Placement	0.8362	SVR	SVM	0.7494			
Credit Card Fraud	0.6490	SVR	SVR	0.6490			

Precision

In this section, the six datasets in the problem space P, have their algorithm performance predicted using the metric precision. As stated in the literature review, precision is the accuracy of positive predictions and there is a trade off of this metric and recall.

Since the algorithms SVR, LR and RR are not normally used for classification, a threshold is necessary to assign class labels. Figure 7 shows that a decision threshold of 0.5 is not optimal for the Credit Card Dataset using SVR. Additional, graphs of the precision and recall versus the decision threshold for the other dataset algorithm combinations are included in Appendix A.

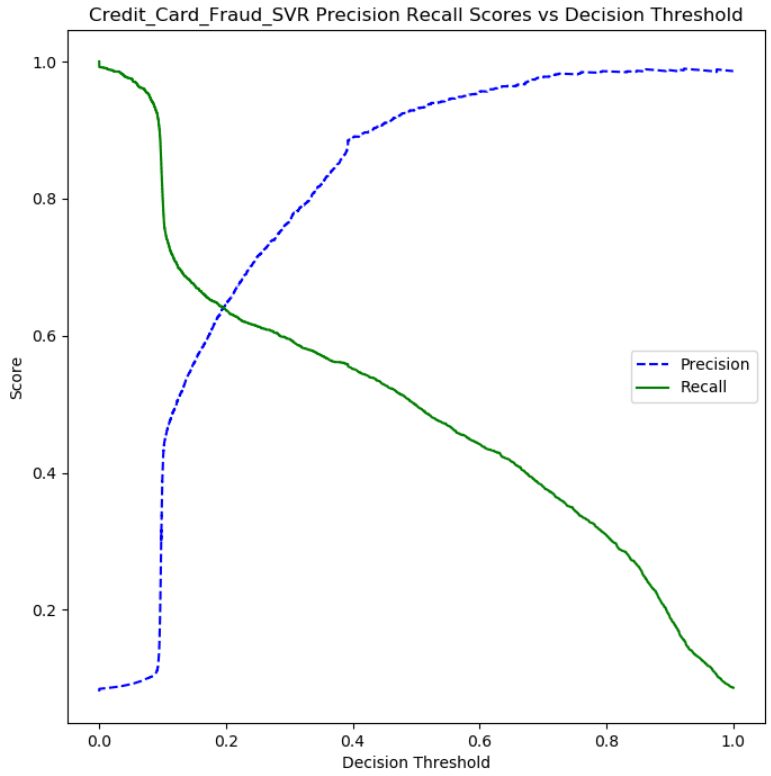

Figure 7. SVR Credit Card Fraud Precision Recall vs Threshold

Table 24 shows each algorithms performance using the performance metric precision. The rankings of each algorithm $a \in A$ is given in Table 25. The regression algorithms used classification true precision ranks in the top three of the algorithms used for 66.67% of the datasets.

Table 24. Dataset Actual Precision

Dataset	SVM	KNN	Naive Bayes	SVR	Ridge Regression	Linear Regression
Heart	0.8689	0.7887	0.8636	0.8485	0.8194	0.8194
Spam	0.8466	0.8804	0.8568	0.8937	0.9457	0.9457
Bank Personal Loan	0.7384	0.9386	0.4806	0.9556	0.9718	0.9718
Framingham	0.2655	0.3333	0.3780	0.5000	0.5000	0.5000
Math Placement	0.8568	0.7627	0.8416	0.7695	0.7569	0.7569
Credit Card Fraud	0.5272	0.8528	0.4587	0.9301	0.9163	0.9068

Table 25. Dataset Actual Precision Rankings

Dataset	SVM	KNN	Naive Bayes	SVR	Ridge Regression	Linear Regression
Heart	1	6	2	3	4.5	4.5
Spam	6	4	5	3	1.5	1.5
Bank Personal Loan	5	4	6	3	1.5	1.5
Framingham	6	5	4	2	2	2
Math Placement	1	4	2	3	5.5	5.5
Credit Card Fraud	5	4	6	1	2	3

Like the previous performance metrics, to predict precision, SVR with a radial basis function kernel is the meta learner with the meta features projected to 3-dimension space as the input and each algorithms' precision as the target variable. Leave one out validation gives the predicted precision of the dataset.

The final predicted precision of the meta models is given in Table 26 and algorithm ranking is given in Table 27.

Table 26. Dataset Predicted Precision

Dataset	SVM	KNN	Naive Bayes	SVR	Ridge Regression	Linear Regression
Heart	0.7359	0.5616	0.6238	0.5676	0.5368	0.5359
Spam	0.6480	0.9286	0.5260	0.9406	0.9621	0.9621
Bank Personal Loan	0.7261	0.6590	0.7126	0.6301	0.6404	0.6393
Framingham	0.7659	0.8507	0.7256	0.8625	0.8585	0.8585
Math Placement	0.6759	0.7244	0.5783	0.7641	0.7563	0.7563
Credit Card Fraud	0.7432	0.7751	0.7037	0.8131	0.8186	0.8186

Table 27. Dataset Predicted Precision Ranking

Dataset	SVM	KNN	Naive Bayes	SVR	Ridge Regression	Linear Regression
Heart	1	4	2	3	5	6
Spam	5	4	6	3	1.5	1.5
Bank Personal Loan	1	3	2	6	4	5
Framingham	5	4	6	1	2.5	2.5
Math Placement	5	4	6	1	2.5	2.5
Credit Card Fraud	5	4	6	3	1.5	1.5

The evaluation of the meta learning recommendation system using precision as a performance metric is given in Table 28. Using precision as the performance metric with classification problems causes the system to recommend the usage of the actual best performing algorithm in half of the datasets used for this thesis. Relaxing the criteria for a hit does not improve the hit ratio.

Table 28. Precision Recommendation Rating

					Epsilon		
Model	Actual Best Precision	Best Algorithm	Recommended Algorithm	Precision of Recommendation	0.01	0.05	0.1
Heart	0.8689	SVM	SVM	0.8689			
Spam	0.9457	RR	RR	0.9457			
Bank Personal Loan	0.9718	RR	SVM	0.7384			
Framingham	0.5000	SVR	SVR	0.5000			
Math Placement	0.8568	SVM	SVR	0.7695			
Credit Card Fraud	0.9301	SVR	RR	0.9301			

Recall

In this section, the six datasets in the problem space P, have their algorithm performance predicted using the metric recall. Recall is the ratio of positive instances that are correctly detected by the classifier. It is also called true positive rate or sensitivity. A classifier with high recall but low precision will have many predicted labels that are incorrect when compared to the training labels. This classifier predicts many positives instances. On the other hand, a classifier with high precision but low recall will have many correct predictions when compared to the training labels but the classifier is predicting many negative instances [24]. Note, there is a precision recall trade off. Increasing recall will reduce precision and vice versa [17].

Table 29 shows each algorithms performance using the metric recall. Since recall is the accuracy of the positive predictions, it will low for algorithms that do not predict many positive instances. This is why the recall for the Framingham and Credit Card data sets are much lower for the algorithms SVR, LR and RR. The rankings of each algorithm $a \in A$ is given in Table 30. SVM is the true best performing algorithm in 66.67% of the datasets.

Table 29. Dataset Actual Recall

Dataset	SVM	KNN	Naive Bayes	SVR	Ridge Regression	Linear Regression
Heart	0.8030	0.8485	0.8636	0.8485	0.8939	0.8939
Spam	0.7766	0.7614	0.4538	0.7186	0.4566	0.4566
Bank Personal Loan	0.9115	0.5573	0.6458	0.6719	0.3594	0.3594
Framingham	0.6143	0.0897	0.2152	0.0448	0.0045	0.0045
Math Placement	0.6660	0.8351	0.7010	0.9155	0.8990	0.8990
Credit Card Fraud	0.6945	0.4878	0.5423	0.4984	0.2807	0.2811

Table 30. Dataset Actual Recall Rankings

Dataset	SVM	KNN	Naive Bayes	SVR	Ridge Regression	Linear Regression
Heart	6	4.5	3	4.5	1.5	1.5
Spam	1	2	6	3	4.5	4.5
Bank Personal Loan	1	4	3	2	5.5	5.5
Framingham	1	3	2	4	5.5	5.5
Math Placement	6	4	5	1	2.5	2.5
Credit Card Fraud	1	4	2	3	6	5

Like the previous performance metrics, to predict recall, SVR with a radial basis function kernel is the meta learner with the meta features projected to 3 - dimensional space as the input and each algorithms' recall as the target variable. Like the previous sections, the hyperparameters are set to the defaults. The predicted recall of the dataset that is left out during leave one out validation is the predicted precision of the meta modeler.

The final predicted recall of the meta models is given in Table 31 and algorithm ranking is given in Table 32.

Table 31. Dataset Predicted Recall

Dataset	SVM	KNN	Naive Bayes	SVR	Ridge Regression	Linear Regression
Heart	0.6417	0.6699	0.5991	0.6494	0.4059	0.4061
Spam	0.8879	0.6133	0.5802	0.6246	0.3693	0.3693
Bank Personal Loan	0.7087	0.6807	0.4649	0.6759	0.6603	0.6603
Framingham	0.7708	0.7273	0.7279	0.8113	0.6710	0.6710
Math Placement	0.7208	0.5743	0.6343	0.6368	0.4285	0.4285
Credit Card Fraud	0.7628	0.6978	0.6228	0.7380	0.6033	0.6033

Table 32. Dataset Predicted Recall Rankings

Dataset	SVM	KNN	Naive Bayes	SVR	Ridge Regression	Linear Regression
Heart	3	1	4	2	6	5
Spam	1	3	4	2	6	5
Bank Personal Loan	1	2	6	3	5	4
Framingham	2	4	3	1	6	5
Math Placement	1	4	3	2	6	5
Credit Card Fraud	1	3	4	2	5.5	5.5

Table 33 shows evaluation of the meta learning recommendation system using recall as the performance metric. The meta learning recommendation system recommends the actual best performing algorithm 50% of the time. Additionally, relaxing the tolerance of a hit to being within 10% of the actual best algorithm, improves the hit rate to 66.67%.

Table 33. Recall Recommendation Rating

					Epsilon		
Model	Actual Best Recall	Best Algorithm	Recommended Algorithm	Recall of Recommendation	0.01	0.05	0.1
Heart	0.8939	RR	KNN	0.8485			
Spam	0.7766	SVM	SVM	0.7766			
Bank Personal Loan	0.9115	SVM	SVM	0.9115			
Framingham	0.6143	SVM	SVR	0.0448			
Math Placement	0.9155	SVR	SVM	0.6660			
Credit Card Fraud	0.6945	SVM	SVM	0.6945			

Figure 8 shows recalls versus the time to execute each algorithm for the Credit Card dataset. Since the dataset is large SVR, SVM and KNN had a slow execution time. The algorithm Naive Bayes Classifier dominates RR, LR, KNN and SVR. Naive Bayes does not dominate SVM but there is a practical difference in execution time. SVM had that best recall of 0.6945 but it took 12.45 minutes to execute that algorithm versus Naive Bayes which has a recall of 0.5423 but execute instantaneously on this dataset.

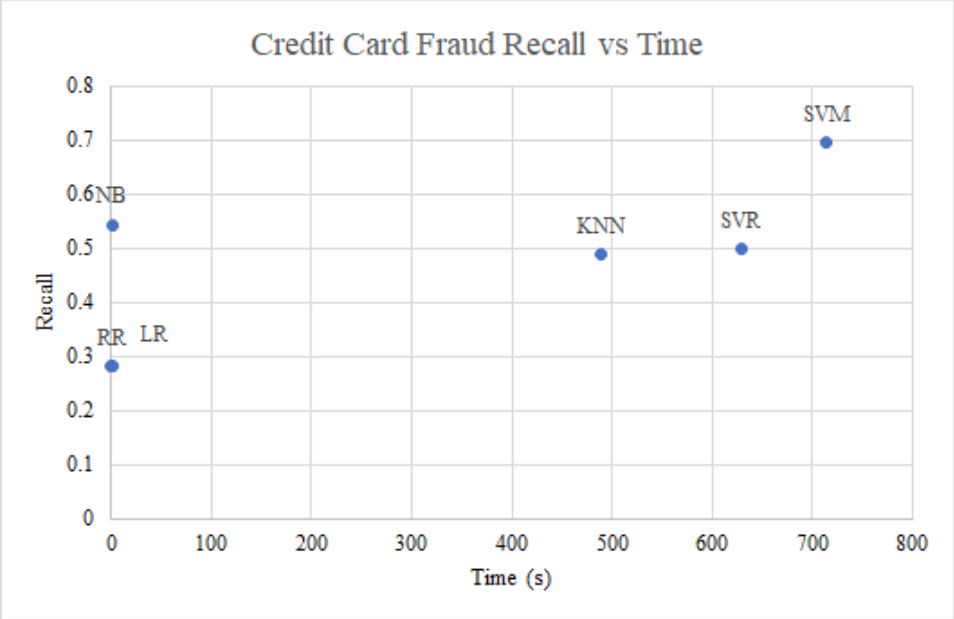

Figure 8. Credit Card Fraud Recall vs Time

Comparison

Table 34 and Table 35 show the actual and predicted algorithm ranking for each dataset when the regression algorithms are removed. The hit rates improves to 66.67%.

Table 34. Recall Classification Algorithms Actual Ranking

Dataset	SVM	KNN	Naive Bayes
Heart	3	2	1
Spam	1	2	3
Bank Personal Loan	1	3	2
Framingham	1	3	2
Math Placement	3	1	2
Credit Card Fraud	1	3	2

Table 35. Recall Classification Algorithms Predicted Ranking

Dataset	SVM	KNN	Naive Bayes
Heart	2	1	3
Spam	1	2	3
Bank Personal Loan	1	2	3
Framingham	1	3	2
Math Placement	1	3	2
Credit Card Fraud	1	2	3

Table 36. Credit Card Fraud Evaluation Metrics Comparison

NRMSE	NRMSE Probabilitiesbilities	Precision	Recall	F1 Score
SVR	SVM	RR	SVM	SVR

Table 36 shows the comparison of the recommendation r' of the Credit Card Fraud dataset for each the member of the performance space z. For this dataset, the recommendation system gives the true best algorithm when NRMSE, Recall and F1 Score are the performance metric used to rank the algorithms. When using NRMSE as the evaluation metric, SVR is the recommended algorithm. This recommendation is correct. However, inspecting the confusion matrix for SVR shown in Table 72 shows that the algorithm is only classifying 49.84% of the fraudulent credit card transactions correctly. The performance metric recall changes the recommendation r' to SVM. In this case, the confusion matrix shown in Table 37, reveals the amount of fraudulent transactions correctly classified improves to 69.45%. Additional, confusion matrices for every dataset, algorithm combination are in Appendix B.

Table 37. Credit Card Fraud: SVM Confusion Matrix

	Predicted	
Truth	Class 1	Class 2
Class 1	26407	1531
Class 2	751	1707

Table 38. Credit Card Fraud: SVR Confusion Matrix

	Predicted	
Truth	Class 1	Class 2
Class 1	27846	92
Class 2	1233	1225

V. Conclusion

Meta Learning is considered using the features of a dataset to develop an overarching model about the features. The usage of meta learning for algorithm selection originates from Rice's model in which the purpose is to select a good or best algorithm for a particular problem [3].

This research shows that there is empirical evidence to suggest that NRMSE or precision or recall could be the performance metric of choice to rank algorithm selection for classification datasets when using the meta learning recommendation system. The meta learning recommendation system is able to recommendation the true best algorithm most frequently when NRMSE is the performance space metric. However, in this case, the performance metric NRMSE favors algorithms the minimize incorrect predictions which can led to a recommendation r' that only considers accuracy. Therefore, the performance metric of choice should be based on whether one wants to accept more Type I error, which is false positives, or Type II error, which is false negatives.

There are multiple areas for future research. First since algorithms that are normally suited for continuous output can perform well for binary classification, separate meta learners could be created for the two differing sets of algorithms. Once these new meta learners are created, they should contain the algorithms that are exclusive for that particular output. For example, a meta learner for classification datasets should only contain algorithms that are suitable for discrete output. This will allow the meta learner to employ functions native to python or any other programming language without overriding standard behavior. Another area that was not explored in this thesis was model adequacy for the algorithms that were ranked. F-tests or other statistical tests can be used to determine if the recommended algorithms are a good fit to the respective dataset. Another area of further research, is to add dif-

ferent machine learning algorithms like neural networks, decision tree and logistic regression, etc to the available algorithms for ranking by the framework. Subsequent research can also use design of experiments to sample datasets of varying complexity to tune the meta learner. Theses sample datasets can be generated using real valued unimodal or multimodal function. Also, different meta features can be explored to predict the absolute expected performance for algorithm recommendation.

Appendix A. Additional Figures

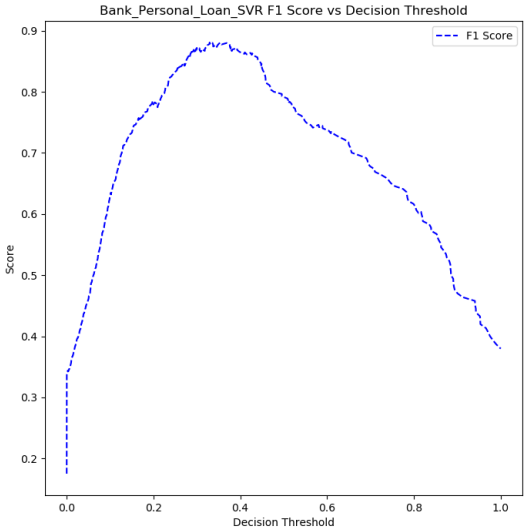

Figure 9. SVR Bank Personal Loan F1 Score vs Decision Threshold

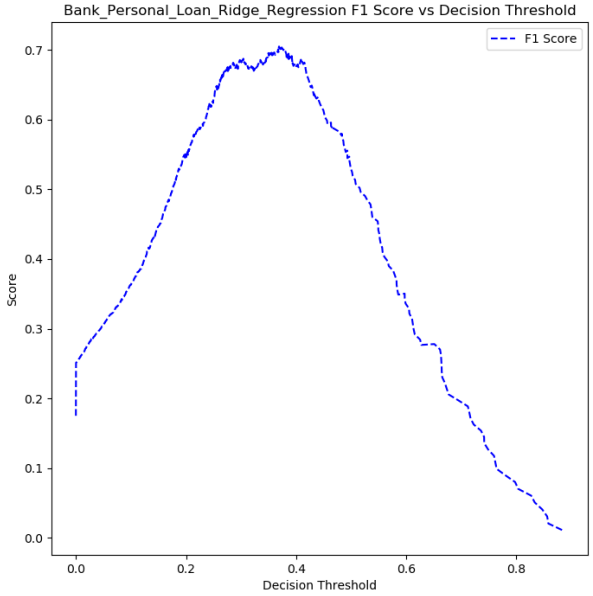

Figure 10. Ridge Regression Bank Personal Loan F1 Score vs Decision Threshold

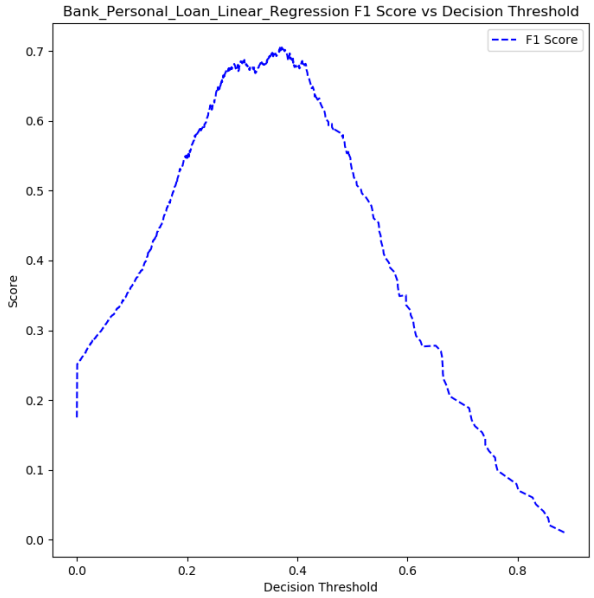

Figure 11. Linear Regression Bank Personal Loan F1 Score vs Decision Threshold

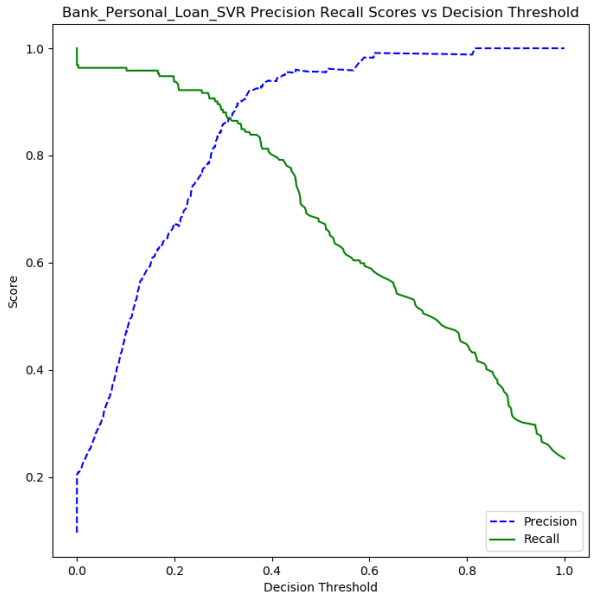

Figure 12. SVR Bank Personal Loan Precision/Recall vs Decision Threshold

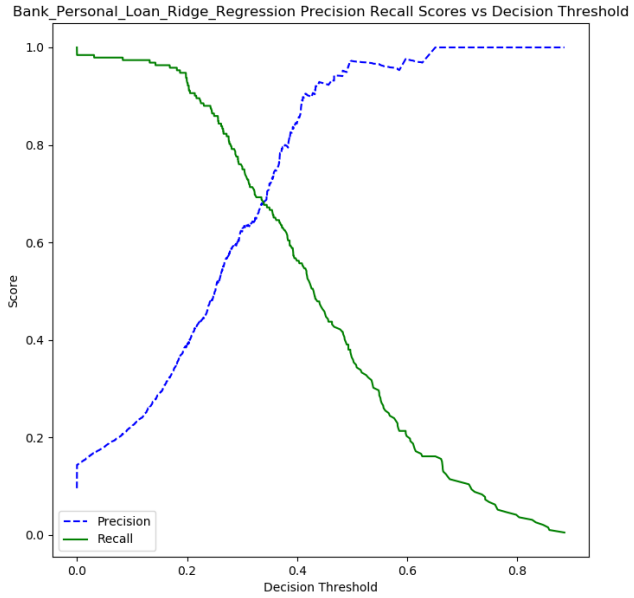

Figure 13. RR Bank Personal Loan Precision/Recall vs Decision Threshold

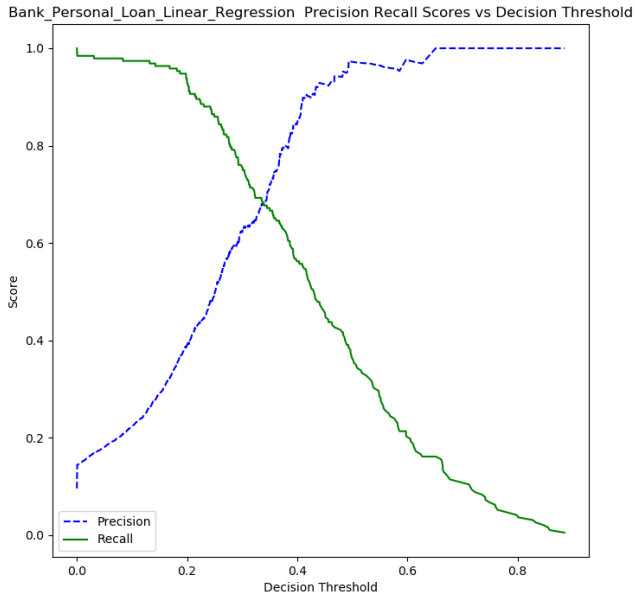

Figure 14. LR Bank Personal Loan Precision/Recall vs Decision Threshold

Appendix B. Confusion Matrices

Table 39. Heart: SVM Confusion Matrix

	Predicted	
Truth	Class 1	Class 2
Class 1	48	8
Class 2	13	53

Table 40. Heart: KNN Confusion Matrix

	Predicted	
Truth	Class 1	Class 2
Class 1	41	15
Class 2	10	56

Table 41. Heart: Naive Bayes Confusion Matrix

	Predicted	
Truth	Class 1	Class 2
Class 1	47	9
Class 2	9	57

Table 42. Heart: SVR Confusion Matrix

	Predicted	
Truth	Class 1	Class 2
Class 1	46	10
Class 2	10	56

Table 43. Heart: Ridge Regression Confusion Matrix

	Predicted	
Truth	Class 1	Class 2
Class 1	43	13
Class 2	7	59

Table 44. Heart: Linear Regression Confusion Matrix

	Predicted	
Truth	Class 1	Class 2
Class 1	43	13
Class 2	7	59

Table 45. Spam: SVM Confusion Matrix

	Predicted	
Truth	Class 1	Class 2
Class 1	1014	102
Class 2	162	563

Table 46. Spam: KNN Confusion Matrix

	Predicted	
Truth	Class 1	Class 2
Class 1	1041	75
Class 2	173	552

Table 47. Spam: NB Confusion Matrix

	Predicted	
Truth	Class 1	Class 2
Class 1	1061	55
Class 2	396	329

Table 48. Spam: SVR Confusion Matrix

	Predicted	
Truth	Class 1	Class 2
Class 1	1054	62
Class 2	204	521

Table 49. Spam: Ridge Regression Confusion Matrix

	Predicted	
Truth	Class 1	Class 2
Class 1	1097	19
Class 2	394	331

Table 50. Spam: Linear Regression Confusion Matrix

	Predicted	
Truth	Class 1	Class 2
Class 1	1097	19
Class 2	394	331

Table 51. Bank: SVM Confusion Matrix

	Predicted	
Truth	Class 1	Class 2
Class 1	1746	62
Class 2	17	175

Table 52. Bank: KNN Confusion Matrix

	Predicted	
Truth	Class 1	Class 2
Class 1	1801	7
Class 2	85	107

Table 53. Bank Personal Loan: Naive Bayes Confusion Matrix

	Predicted	
Truth	Class 1	Class 2
Class 1	1674	134
Class 2	68	124

Table 54. Bank Personal Loan: SVR Confusion Matrix

	Predicted	
Truth	Class 1	Class 2
Class 1	1802	6
Class 2	63	129

Table 55. Bank Personal Loan: Ridge Regression Confusion Matrix

	Predicted	
Truth	Class 1	Class 2
Class 1	1806	2
Class 2	123	69

Table 56. Bank Personal Loan: Linear Regression Confusion Matrix

	Predicted	
Truth	Class 1	Class 2
Class 1	1806	2
Class 2	123	69

Table 57. Framingham: SVM Confusion Matrix

	Predicted	
Truth	Class 1	Class 2
Class 1	861	379
Class 2	86	137

Table 58. Framingham: KNN Confusion Matrix

	Predicted	
Truth	Class 1	Class 2
Class 1	1200	40
Class 2	203	20

Table 59. Framingham: Naive Bayes Confusion Matrix

	Predicted	
Truth	Class 1	Class 2
Class 1	1161	79
Class 2	175	48

Table 60. Framingham: SVR Confusion Matrix

	Predicted	
Truth	Class 1	Class 2
Class 1	1230	10
Class 2	213	10

Table 61. Framingham: Ridge Regression Confusion Matrix

	Predicted	
Truth	Class 1	Class 2
Class 1	1239	1
Class 2	222	1

Table 62. Framingham: Linear Regression Confusion Matrix

	Predicted	
Truth	Class 1	Class 2
Class 1	1239	1
Class 2	222	1

Table 63. Math Placement: SVM Confusion Matrix

	Predicted	
Truth	Class 1	Class 2
Class 1	177	54
Class 2	162	212

Table 64. Math Placement: KNN Confusion Matrix

	Predicted	
Truth	Class 1	Class 2
Class 1	105	126
Class 2	80	405

Table 65. Math Placement: Naive Bayes Confusion Matrix

	Predicted	
Truth	Class 1	Class 2
Class 1	167	64
Class 2	145	340

Table 66. Math Placement: SVR Confusion Matrix

	Predicted	
Truth	Class 1	Class 2
Class 1	98	133
Class 2	41	444

Table 67. Math Placement: Ridge Regression Confusion Matrix

	Predicted	
Truth	Class 1	Class 2
Class 1	91	140
Class 2	49	436

Table 68. Math Placement: Linear Regression Confusion Matrix

	Predicted	
Truth	Class 1	Class 2
Class 1	91	140
Class 2	49	436

Table 69. Credit Card Fraud: KNN Confusion Matrix

	Predicted	
Truth	Class 1	Class 2
Class 1	27731	207
Class 2	1259	1199

Table 70. Credit Card: Naive Bayes Confusion Matrix

	Predicted	
Truth	Class 1	Class 2
Class 1	26365	1573
Class 2	1125	1333

Table 71. Credit Card Fraud: Ridge Regression Confusion Matrix

	Predicted	
Truth	Class 1	Class 2
Class 1	27875	63
Class 2	1768	690

Table 72. Credit Card Fraud: Linear Regression Confusion Matrix

	Predicted	
Truth	Class 1	Class 2
Class 1	27867	71
Class 2	1767	691

Appendix C. Source Code

```
Author Megan Woods
Modified on 22 Feb 2020
by Clarence Williams
Added classification dataset functionality

This script contains inputs that the user will need to change to run
    with
his/her system, as well as options that the user may want to edit.

It is called in
    - _03_prepare_data
    - _04_algorithms
    - _05_main
"""
# Constants

# Directories.....................................
current_dir = "C:\\Users\\c3_wi\\Desktop\\Python"
data_dir = current_dir + "data"
documentation_dir = current_dir

# Current...
#documentation_dir = "C:\\Users\\megan.woods\\Desktop\\Megan Woods\\
    AFIT\\Thesis\\recommendation_tool\\2019.07.07_start\\
    documentation/"

# Don't touch.....................................
sigDig = 4 # number of significant digits
min_rows = 7 # minimum number of rows a dataset must contain
```

Listing C.1. Constants.py

```
This script contains helper functions. Placed here to declutter other scripts.

It is called in
    - _04_algorithms
    - _05_main"""

# Helper functions for the recommendation system

def unique(list1):
    # intilize a null list
    unique_list = []
    # traverse for all elements
    for x in list1:
        # check if exists in unique_list or not
        if x not in unique_list:
            unique_list.append(x)
    return unique_list

def find_ranks(performance_dict, return_sorted = False):
    """ Function to find rankings of the algorithms

        Parameters
        ---------
            performance_dict: dictionary
                performances calculated per algorithm
            return_sorted = boolean
                False: return ranks ordered by order of algorithms in calculate_accuracies function
                True: return ranks ordered from highest to lowest

        Returns
```

```
31          -------
32              dictionary, where keys are algorithms and values are
    ranks
33      """
34      perf = performance_dict.copy()
35      ranks_dict = {key: rank for rank, key in enumerate(sorted(set(
    perf.values()), reverse=True), 1)}
36      ranks = {k: ranks_dict[v] for k,v, in perf.items()} # unordered
    ranks
37
38      if return_sorted == True:
39          num = 1
40          ranks_ordered = {}
41          ranks_temp = ranks.copy()
42          while num != len(ranks)+1:
43              h_rank = min(ranks_temp.items(), key=lambda x: x[1]) #
    find key, value with highest rank
44              ranks_ordered[h_rank[0]] = h_rank[1] # add key, value to
     new dictionary
45              ranks_temp.pop(h_rank[0]) # remove key, value from temp
    dictionary
46              num = num + 1 # update indicator
47          return ranks_ordered
48      else:
49          return ranks
50
51 # OTHER_____________________________
52 # ___________________________________
53 def extract(myDict, keys = [], values = []):
54      """ Function to get a subset of dictionary from a dictionary
55
56          Parameters
```

```
        ----------
            myDict: dict
                the dictionary from which to extract
            keys: list
                names of keys to subset on
            values: list
                values to search for

        Returns
        -------
            subset of dictionary
    """
    if len(values) != 0:
        return dict((k, myDict[k]) for k, v in myDict.items() if v
   in values)
    if len(keys) != 0:
        return dict((k, myDict[k]) for k in keys if k in myDict)

def is_number(s):
    """ checks to see if data in file is a number or not
    """
    try:
        float(s)
        return True
    except ValueError:
        pass
    try:
        import unicodedata
        unicodedata.numeric(s)
        return True
    except (TypeError, ValueError):
        pass
```

```
88      return False
89
90  def maybe_float(s):
91      try:
92          return int(s)
93      except (ValueError, TypeError):
94          return s
95
96  # find all values in df that are in datasets
97  def intersection(lst1, lst2):
98      temp = set(lst2)
99      lst3 = [value for value in lst1 if value in temp]
100     return lst3
```

Listing C.2. my_functions.py

```
# -*- coding: utf-8 -*-
"""
Created on 01 March 2019
Author Megan Woods
Modified on 22 Feb 2020
by Clarence Williams
Added classification dataset functionality

This script is used to set up a single dataset. It
    - Loads the dataset
    - Preprocesses
    - Determines target column
    - Creates training and testing sets
    - Finds meta-features

It is called in
    - _05_main

"""
import os
import numpy as np
import pandas as pd

from pydataset import data as pydata
from sklearn.model_selection import train_test_split
from sklearn import preprocessing
from sklearn.preprocessing import StandardScaler

import _01_constants as constants

# Prepare the data_________
# ************************
```

```
# ************************
class Prep_Data():
    """ For setting up a new data instance

        Parameters
        ----------
            name: string
                a string to identify this dataset
            target: string
                name of target column to predict
            directory: string
                location of csv file
            csv_file: string
                name of data to load
            dataframe: dataframe
                already loaded dataframe to pass in (optional)
            train_test: binary
                whether or not to split into training and testings
    sets

    """
    def __init__(self, name, target_col="", directory="", csv="",
    dataframe="", train_test=True):
        # init functions_____________________
        # ___________________________________

        def compute_test_train(target, df):
            """ Function that computes the train and test datasets
            """
            sc = StandardScaler()
            x_df = df.drop(target,1)
            y_df = df[target]
```

```
63            x_df_scaled = sc.fit_transform(x_df)
64            X_train, X_test, Y_train, Y_test = train_test_split(
    x_df_scaled, y_df, test_size=0.4, random_state=1, stratify=y_df)
65            return X_train, X_test, Y_train, Y_test
66
67        def read_documentation():
68            os.chdir(constants.documentation_dir)
69            filename = name+".txt"
70            if os.path.isfile(filename):
71                file = open(filename, "r", encoding="utf-8")
72                contents = file.read()
73                file.close()
74            else:
75                contents = "DNE"
76            os.chdir(constants.current_dir)
77            return contents
78
79        def determine_int_vs_float(df):
80            float_col_list = []
81            int_col_list = []
82
83            for col in df.columns:
84                if all(isinstance(x,float) for x in df[col])==True:
85                    float_col_list.append(col)
86                elif all(isinstance(x,int) for x in df[col])==True:
87                    int_col_list.append(col)
88            float_df = df[float_col_list]
89            int_df = df[int_col_list]
90            return float_df, int_df
91
92        def separate_cont_and_discrete(df):
93
```

```
94            """Question: How do we know if the data is discrete?
95                12 int64 and 1 float64 column
96
97                number of unique values: 228, 9, 97, 120, 108, 98,
    82, 56, 195, 204, 143, 172, 10
98                for  i in temp.columns: print(238/temp[i].nunique())
99                    1  26  2.5  2  2.2  2.4  2.9  4.3  1  1  1.6
    1.4  23
100
101           """
102           sub_df = df.select_dtypes(include=["number"])
103           for i in sub_df.columns:
104               if i == "idp":
105                   print(i)
106
107           sub_df = sub_df.drop(self.target,1).copy()
108           num_rows = len(sub_df)
109           indicator = int(num_rows<30)
110           threshold = 0.25*(1-indicator) + 0.5*indicator
111           #threshold = 0.25 if numRows >= 30, 0.5 if numRows < 30
112
113           # current method to determine if discrete:
114           discrete = []
115           for i in sub_df.columns:
116               value = "Continuous"
117               if sub_df[i].nunique()/num_rows <= threshold:
118                   value = "Discrete"
119                   discrete.append(i)
120
121           discrete_df = sub_df[[x for x in sub_df.columns if x in
    discrete]]
122           continuous_df = sub_df[[x for x in sub_df.columns if x
```

```
    not in discrete]]
123
124             return continuous_df, discrete_df
125
126         def set_target(df):
127             """ Set the target, unless already specified
128             """
129             number_list = list(df.select_dtypes(include=["number"]).
    columns)
130             unique_values = list(df[number_list].nunique())
131             max_value = max(unique_values)
132             idx = unique_values.index(max_value)
133             target = number_list[idx]
134             return target
135
136         # init variables_______________________________
137
138         self.name = name
139         self.documentation = read_documentation()
140         self.remove = False
141
142         # Load data
143         if directory is not "":
144             temp_df = pd.read_csv("C:\\Users\\c3_wi\\Desktop\\Python
    \\Data\\" + csv)
145         else:
146             temp_df = pydata(name)
147         # Preprocessing_____________________________________
148         temp_df = temp_df.dropna(1,how="all").dropna(0,how="any")
149
150
151
```

```
152         cols = temp_df.columns.copy()
153         for i in cols:
154             # if column is boolean
155             if temp_df[i].dtype.name=="bool":
156                 # change values to 0 and 1
157                 temp_df[i] = temp_df[i].astype(int)
158             # drop columns that have the exact same input for each
    row
159             if temp_df[i].nunique() == 1:
160                 temp_df = temp_df.drop(i,1)
161             # drop columns that serve as an index column
162             elif list(temp_df.index) == list(temp_df[i]):
163                 temp_df = temp_df.drop(i,1)
164 
165         # drop object columns that have all unique values
166         object_cols = temp_df.select_dtypes(include=["object"]).
    columns
167         df = temp_df.drop((i for i in object_cols if len(temp_df[i].
    unique())==len(temp_df[i])),1)
168 
169         # we need at least 3 numeric columns (including the target
    column) in order to take a gradient
170         temp_num_cols = temp_df.select_dtypes(include=["number"]).
    columns
171         if len(temp_num_cols) <= 2:
172             self.remove = True
173         elif len(temp_df) < constants.min_rows:
174             self.remove = True
175 #         elif len(temp_df.columns) < min_cols:
176 #             self.remove = True
177         else:
178             num_columns_in_modified_original_df = len(df.columns)
```

```
            # set target column
            if (target_col == "") or (target_col not in
    temp_num_cols):
                #self.target = set_target(df)
                self.target = target_col
            else:
                self.target = target_col
            target = self.target
            self.original_data = df.copy()

            # label encode response
            le = preprocessing.LabelEncoder()

            df[str(target)] = le.fit_transform(df[str(target)])

#            continuous_df, discrete_df = determine_int_vs_float(df)
            continuous_df, discrete_df = separate_cont_and_discrete(
    df)
            num_continuous = len(continuous_df.columns)
            num_discrete = len(discrete_df.columns)

            # find greatest number of unique values in discrete
    column
            disc_num_unique = []
            for col in discrete_df.columns:
                disc_num_unique.append(len(discrete_df[col].unique()
    ))

            if len(disc_num_unique) != 0:
                max_disc_num_unique = max(disc_num_unique)
                min_disc_num_unique = min(disc_num_unique)
                avg_disc_num_unique = sum(disc_num_unique)/float(len
```

```
(disc_num_unique))
207             else:
208                 max_disc_num_unique = 0
209                 min_disc_num_unique = 0
210                 avg_disc_num_unique = 0
211 
212             # one hot encoding for categorical variables
213             df = pd.get_dummies(df,drop_first=True) #dtype = "
    float64"
214 
215             if len(continuous_df.columns) != 0:
216                 df[continuous_df.columns] = (pd.DataFrame(#data=
    min_max_scaler.fit_transform(continuous_df),
217                                                           data=
    preprocessing.scale(continuous_df),
218                                                           index=
    continuous_df.index,
219                                                           columns=
    continuous_df.columns))
220 
221             self.numeric_df = df.select_dtypes(include=["number"]).
    drop(target,1)
222             num_df = self.numeric_df
223             self.num_pred_cols = len(num_df.columns)
224 
225             gradient = np.gradient(num_df.values) # do we compute
    the gradient on the numeric and hot-encoded?
226             horizontal_gradient = gradient[1] # differences computed
     per row - pretty sure we want this one
227 
228             meta_features = {
229                                 "Rows": len(df.index),
```

```
                            "Columns":
num_columns_in_modified_original_df,
                            "Rows-Cols Ratio": len(df.index)/
num_columns_in_modified_original_df,
                            "Number Discrete": num_discrete,
                            "Max num factors": max_disc_num_unique,
                            "Min num factors": min_disc_num_unique,
                            "Avg num factors": avg_disc_num_unique,
                            "Number Continuous": num_continuous,
                            "Gradient-Avg": horizontal_gradient.
mean(),
                            "Gradient-Min": horizontal_gradient.min
(),
                            "Gradient-Max": horizontal_gradient.max
(),
                            "Gradient-Std": horizontal_gradient.std
()
                    }

          meta_features = pd.DataFrame(data=[[v for v in
meta_features.values()]],
                                       columns=[k for k in
meta_features.keys()],
                                       index=[name])
          self.meta_features = meta_features

          # Training
          if train_test:
              self.X_train, self.X_test, self.y_train, self.y_test
 = compute_test_train(df=df,target=target)
              self.num_train = len(self.X_train)
```

```
252                 self.num_test   = len(self.X_test)
```

Listing C.3. prepare_data.py

```
1  # -*- coding: utf-8 -*-
2  """
3  Author Megan Woods
4  Modified on 22 Feb 2020
5  by Clarence Williams
6  Added classification dataset functionality
7
8  - performs regression for algorithms
9
10 It is called in
11     - _05_main
12     - app
13 """
14 import _01_constants as constants
15 import _02_my_functions as mf
16
17 import os
18 import numpy as np
19 from sklearn import linear_model, neighbors, tree
20 from sklearn.metrics import mean_squared_error, r2_score
21 from sklearn import svm
22 from sklearn.neighbors import KNeighborsClassifier
23 from sklearn.naive_bayes import GaussianNB
24 from sklearn.metrics import precision_score, recall_score
25 from sklearn.ensemble import RandomForestClassifier
26 from sklearn.tree import DecisionTreeClassifier
27
28
29 os.chdir(constants.current_dir)
```

```
30
31 # MODELS_____________________
32 models = {
33           "SVM":  svm.SVC(kernel='rbf', probability=False, gamma='
       scale', class_weight='balanced'),
34           "KNN": KNeighborsClassifier(n_neighbors=5),
35            "NB": GaussianNB(),
36           }
37
38 alg_names = list(models.keys())
39 algs_to_scale = ["SVM", "KNN", "NB"]
40 regression_num = 1 # to differentiate regression plots
41
42 class Metamodel():
43     """ Class to create a (linear or decision tree) regression model
       . Can plot
44         the predicted vs. actual plots, residual plots, as well as
       display
45         statistical performances of the model
46     """
47     def __init__(self, reg_type, data):
48         self.data = data
49         self.target = data.target
50         self.name = reg_type
51         self.model = models[reg_type]
52         self.model.fit(data.X_train, data.y_train)
53         self.pred_train = self.model.predict(data.X_train)
54         self.pred_test = self.model.predict(data.X_test)
55         self.recall = recall_score(data.y_test, self.pred_test,
       average='weighted')
56
57
```

```
58      def predict_y(self, point):
59          self.predicted_point = self.model.predict(point)
60
61  class Algorithms_Results():
62      def __init__(self, data):
63
64          ##### Step 1 #####
65          # feature reduction
66
67          ##### Step 2 #####
68          # run regression on dataset
69          models = {}
70          for i in alg_names:
71              models[i] = Metamodel(i, data)
72
73          ##### Step 3 #####
74          # find recall performances for each algorithm
75          algorithm_results = [models[i] for i in models]
76
77          performances_recall = {}
78
79          for i in algorithm_results:
80              performances_recall[i.name] = i.recall
81
82
83          ##### Step 4 #####
84          # rank the algorithms based on their recall performances
85          ranks = mf.find_ranks(performances_recall)
86          ranks_ordered = mf.find_ranks(performances_recall,
    return_sorted=True)
87
88          self.models = models
```

```
89          self.name = data.name
90          self.target = data.target
91          self.num_cols = data.num_pred_cols
92          self.num_train = data.num_train
93          self.num_test = data.num_test
94          self.performances_recall = performances_recall
95          self.ranks = ranks
96          self.ranks_ordered = ranks_ordered
```

Listing C.4. algorithms.py

```
# -*- coding: utf-8 -*-
"""
Author Megan Woods
Modified on 22 Feb 2020
by Clarence Williams
Added classification dataset functionality

It is called in
    - app
"""
# import packages_________________________________
import _01_constants as constants
import _02_my_functions as mf
import _03_prepare_data as prep
import _04_algorithms as algs

import os
import pandas as pd
import numpy as np
from operator import sub, truediv
from sklearn.linear_model import LinearRegression
from scipy.stats import spearmanr
from pydataset import data as pydata
from sklearn.svm import SVR


os.chdir(constants.current_dir)

# SETUP___________________________________

def get_meta_features(my_list):
    """ Creates dataframe of datasets and their metafeature values
```

```
    """
    return pd.concat([i.meta_features for i in my_list])


# def alg_rankings(my_list):
#     """ Creates dataframe of datasets and their rankings for each meta-model
#     """
#     data = [algs.Algorithms_Results(my_list[i]).ranks.values() for i in range(len(my_list))]
#     alg_results = pd.DataFrame(data=data, columns=algs.models.keys(), index=dataset_names)
#     return alg_results

def meta_model(combined_meta, metric_df, target_set, algorithm_name):

    idx = metric_df.columns.get_loc(algorithm_name)
    m, n = combined_meta.shape
    #pca = PCA(n_components=3)
    #y_pred = np.zeros(shape=(m, 1))
    #pca_train_data = pca.fit_transform(X_train)
    #pca_test_data = pca.transform(X_test)
    #y_train, y_test = metric_df.iloc[train_index, idx], metric_df.iloc[test_index, idx]
    idx = metric_df.columns.get_loc(algorithm_name)
    y_train = metric_df.iloc[:, idx]
    # Calculate actual gamma values to test
    model = SVR(C=1, epsilon=0.1, gamma='scale')

    # model = linear_model.LinearRegression()
    model.fit(combined_meta, y_train)
```

```
60
61      y_pred = model.predict(target_set)
62
63      return y_pred
64
65
66
67
68  def rmse_results(my_list):
69      """ Creates dataframe of datasets and their rmse values for each
        meta-model
70      """
71      data = [algs.Algorithms_Results(my_list[i]).performances_rmse.
        values() for i in range(len(my_list))]
72      rmse_results = pd.DataFrame(data=data, columns=algs.models.keys
        (), index=dataset_names)
73      return rmse_results
74
75  def recall_results(my_list):
76      """ Creates dataframe of datasets and their rmse values for each
        meta-model
77      """
78      data = [algs.Algorithms_Results(my_list[i]).performances_recall.
        values() for i in range(len(my_list))]
79      recall_results = pd.DataFrame(data=data, columns=algs.models.
        keys(), index=dataset_names)
80      return recall_results
81
82
83  def get_normalizer(my_list):
84      """
85      """
```

```
    my_max = [i.y_train.max() for i in datasets]
    my_min = [i.y_train.min() for i in datasets]

    range_list = list(map(sub, my_max, my_min))
    return range_list


valid_datasets = []
# CSV Datasets
d1 = ["heart", constants.data_dir, 'heart.csv', "target"]
d2 = ["spam", constants.data_dir, 'spam7.csv', 'yesno_bin']
d3 = ["bank_personal_loan", constants.data_dir, 'Bank_Personal_Loan2
    .csv', 'Personal Loan']
d4 = ["framingham", constants.data_dir, 'framingham2.csv', '
    TenYearCHD']
d5 = ['math_placement', constants.data_dir, 'math_placement3.csv', '
    CourseSuccess']
d6 = ['Credit_Card_Fraud', constants.data_dir, '699.csv', 'isFraud']

potential_datasets_CSVs = [d1, d2, d3, d4, d5, d6]

# run CSV datasets through PrepData
for i in potential_datasets_CSVs[:30]:
    prepared_data = prep.Prep_Data(name=i[0], directory=i[1], csv=i
    [2], target_col=i[3])
    print(i)
    if prepared_data.remove == False:
        valid_datasets.append(prepared_data)


# set variables_____________________________
datasets = valid_datasets  # list of instances of Prep_Data
```

```
dataset_names = [datasets[i].name for i in range(len(datasets))]  # list of dataset names
meta_features = get_meta_features(datasets)
recall_values = recall_results(datasets)
range_list = get_normalizer(datasets)
alg_class = [algs.Algorithms_Results(datasets[i]) for i in range(len(datasets))]



excel_data = pd.concat([meta_features, recall_values], axis=1)
excel_data.to_csv(constants.current_dir + "recall.csv")

spearman_correlation = []
relative_performance = []  # higher is better
temp_list = []
subset_rel_perf = []

num_rel_perf_equal_0 = 0
diff_rel_perf_equal_0 = []
diff_rel_perf_equal_0_names = []

num_correct = 0


def recommend(my_target_dataset):
    global num_correct
    global run
    global num_rel_perf_equal_0

    # Target dataset
    target_dataset = datasets[my_target_dataset].name  # set the
```

```
    target dataset
144
145     target_meta_features = [meta_features.loc[target_dataset].values
    ]
146     target_data = meta_features.loc[[target_dataset]] # target's
    meta_features
147     target_actual_recall = recall_values.loc[target_dataset]
148
149     # training datasets
150     meta_X_train = meta_features.drop(target_dataset, 0)
151     meta_y_train = recall_values.drop(target_dataset, 0)
152
153     # Build SVR model
154     svm_pred = meta_model(meta_X_train, meta_y_train, target_data, '
    SVM')
155     knn_pred = meta_model(meta_X_train, meta_y_train,
    target_meta_features, 'KNN')
156     nb_pred = meta_model(meta_X_train, meta_y_train,
    target_meta_features, 'NB')
157     dt_pred = meta_model(meta_X_train, meta_y_train,
    target_meta_features, 'DT')
158     rf_pred = meta_model(meta_X_train, meta_y_train,
    target_meta_features, 'RF')
159
160     frames_pred = [svm_pred[0], knn_pred[0], nb_pred[0], dt_pred[0],
     rf_pred[0]]
161     #combined_pred = pd.concat(frames_pred, axis=1)
162
163     # Make recall predictions
164     target_predicted_recall = frames_pred
165     #target_predicted_recall = combined_pred[0].tolist()
166
```

```
167     recall_zippedlist = list(zip(target_actual_recall,
    target_predicted_recall))
168     recall_comparisons = pd.DataFrame(recall_zippedlist,
169                                       columns=["Actual Recall", "
    Predicted Recall"],
170                                       index=recall_values.columns)
171
172
173     # Results
174     actual_best = recall_comparisons["Actual Recall"].idxmax()
175     predicted_best = recall_comparisons["Predicted Recall"].idxmax()
176     actual_best_recall = recall_comparisons["Actual Recall"].max()
177     predicted_best_actual_recall = recall_comparisons.loc[
    predicted_best, "Actual Recall"]
178
179
180     temp_relative_performance = actual_best_recall /
    predicted_best_actual_recall
181
182     if temp_relative_performance == 0:
183         num_rel_perf_equal_0 += 1
184         diff_rel_perf_equal_0.append(predicted_best_actual_recall -
    actual_best_recall)
185         diff_rel_perf_equal_0_names.append(target_dataset)
186
187     if predicted_best_actual_recall == 0:
188         relative_performance.append(1)
189         subset_rel_perf.append(1)
190     else:
191         relative_performance.append(temp_relative_performance)
192         if temp_relative_performance != 0:
193             subset_rel_perf.append(temp_relative_performance)
```

```
194
195
196     temp_relative_performance = actual_best_recall /
        predicted_best_actual_recall
197
198     relative_performance.append(temp_relative_performance)
199     if temp_relative_performance != 0:
200         subset_rel_perf.append(temp_relative_performance)
201
202         print("\n")
203         print(target_dataset + " Actual best ", actual_best)
204         print("Predicted best ", predicted_best)
205         print("Actual best recall ", actual_best_recall)
206         print("Predicted best actual recall ",
        predicted_best_actual_recall)
207         print("\n")
208
209     if actual_best == predicted_best:
210         num_correct = num_correct + 1
211
212
213 DATASETS = len(datasets)
214 ITERATIONS = 1  # no point in changing this...
215 TOTAL_RUNS = DATASETS * ITERATIONS
216
217 for j in range(ITERATIONS):
218     #     print(j)
219     for i in range(DATASETS):
220         #         print(i)
221         recommend(i)
222
223 print("\n")
```

```
224 print("Accuracy: ", num_correct / TOTAL_RUNS)
225 print("Number Correct:", num_correct)
226 print("Runs:", TOTAL_RUNS)
227 print("Realtive performance: ", sum(relative_performance) /
       TOTAL_RUNS)
228 print("Spearman's Rank: ", sum(spearman_correlation) / TOTAL_RUNS)
229
230
231 print("Subset rel performance", sum(subset_rel_perf) / len(
       subset_rel_perf))
```

Listing C.5. main.py

Bibliography

1. Mark A. Gallagher and Donald L. Allen, "75 Years (1942 through 2017) of Operations Research in the United States Air Force," *Military Operations Research Society*, vol. 22, no. 4, pp. 5–16, 2017.

2. James J. Cochran, *Informs Analytics Body of Knowledge*, Wiley, Hoboken, NJ, 2019.

3. John Rice, "The Algorithm Selection Problem," *Purdue University, Department of Computer Science Technical Reports, Paper 99*, 1975.

4. "A Meta-Learning Assistant for Providing User Support in Machine Learning and Data Mining," Tech. Rep. 26.357, Esprit, 2001.

5. Marin Matija, Johan A.K. Suykens, and Slavko Krajcar, "Load Forecasting using a Multivariate Meta-Learning System," *Expert Systems with Applications*, vol. 40, no. 11, pp. 4427 - 4437, 2013.

6. Can Cui, Teresa Wu, Mengqi Hu, Jeffery D. Weir, and Xiwang Li, "Short-term Building Energy Model Recommendation System: A Meta-Learning Approach," *Applied Energy*, vol. 172, pp. 251–263, 2016.

7. Bernhard Pfahringer and Quan Sun, "Pairwise Meta-Rules for Better Meta-Learning-Based Algorithm Ranking," *Mach Learn*, vol. 93, pp. 141–161, 2013.

8. Can Cui, Mengqi Hu, Jeffery D. Weir, and Teresa Wu, "A Recommendation System for Meta-Modeling: A Meta-Learning based Approach," *Expert Systems with Applications*, vol. 46, pp. 33–44, 2016.

9. Ciro Castiello, Giovanna Castellano, and Anna Fanelli, "Meta-data: Characterization of Input Features for Meta-learning," 07 2005, MDAI, Modeling Decisions for Artificial Intelligence, Second International Conference, pp. 457–468.

10. Lee J. Bain and Max Engelhardt, *Introduction to Probability and Mathematical Statistics*, Duxbury, Pacific Grove, CA, 1992.

11. Jerome H. Friedman, Robert Tibshirani, and Trevor Hastie, *Elements of Statistical Learning*, Springer, 2017.

12. Evelyn Fix and J. L. Hodges, "Discriminatory Analysis. Nonparametric Discrimination: Consistency Properties," *International Statistical Review / Revue Internationale de Statistique*, vol. 57, no. 3, pp. 238–247, 1989.

13. Vahid Mirjalili and Sebastian Raschka, *Python Machine Learning*, Packt Publishing, Birmingham, UK, 2017.

14. Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani, *An Introduction to Statistical Learning with Applications in R*, Springer, 2017.

15. Peter Hall, Byeong U. Park, and Richard J. Samworth, "Choice of Neighbor Order in Nearest-Neighbor Classification," *The Annals of Statistics*, vol. 36, no. 5, pp. 2135–2152, 2008.

16. Bernhard E. Boser, Isabelle M. Guyon, and Vladimir N. Vapnik, "A Training Algorithm for Optimal Margin Classifiers," in *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, New York, NY, 1992, COLT '92, pp. 144–152, ACM.

17. Aurelien Geron, *Hands-On Machine Learning with Scikit-Learn & TensorFlow*, O'Reilly, Sebastopol, California, 2017.

18. Harris Drucker, Christopher J. C. Burges, Linda Kaufman, Alex J. Smola, and Vladimir Vapnik, "Support Vector Regression Machines," in *Advances in Neural Information Processing Systems 9*, pp. 155–161. MIT Press, 1997.

19. Stella M. Clarke, Jan H. Griebsch, and Timothy W. Simpson, "Analysis of Support Vector Regression for Approximation of Complex Engineering Analyses," 09 2003, vol. Volume 2: 29th Design Automation Conference, Parts A and B of *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, pp. 535–543.

20. "Scikit Learn Naive Bayes," World Wide Web Page, Available at *https://scikit-learn.org/stable/modules/naive_bayes.html*, Accessed 06 August 2019.

21. Karl Pearson, "On Lines and Planes of Closest Fit to Systems of Points in Space," *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 2, no. 11, pp. 559–572, 1901.

22. David C. Lay, *Linear Algebra and its Applications*, Pearson, Boston, MA, 2006.

23. "Scikit Learn Confusion Matrix," World Wide Web Page, Available at *https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html*, Accessed 06 August 2019.

24. "Scikit Learn Precision Recall," World Wide Web Page, Available at *https://scikit-learn.org/stable/auto_examples/model_selection/plot_precision_recall.html*, Accessed 03 September 2019.

25. "Heart Disease UCI," World Wide Web Page, Available at *https://www.kaggle.com/ronitf/heart-disease-uci*, Accessed 03 August 2019.

26. "Spam 7," World Wide Web Page, Available at *http://vincentarelbundock.github.io/Rdatasets/doc/DAAG/spam7.html*, Accessed 03 August 2019.

27. "Bank Loan Modeling," World Wide Web Page, Available at *https://www.kaggle.com/itsmesunil/bank-loan-modelling/version/1*, Accessed 03 August 2019.

28. "Logistic Regression to Predict Heart Disease," World Wide Web Page, Available at *https://www.kaggle.com/dileep070/heart-disease-prediction-using-logistic-regression#framingham.csv*, Accessed 03 August 2019.

29. "Math Placement Exam Results," World Wide Web Page, Available at *http://vincentarelbundock.github.io/Rdatasets/doc/Stat2Data/MathPlacement.html*, Accessed 03 August 2019.

30. "Data Description (Details and Discussion)," World Wide Web Page, Available at *https://www.kaggle.com/c/ieee-fraud-detection/discussion/101203#latest-592110*, Accessed 04 August 2019.

31. Megan Woods, "A Metamodel Recommendation System using Meta-Learning," M.S. thesis, AFIT-ENS-MS-20-M-182, School of Engineering and Management, Air Force Institute of Technology (AU), Wright-Patterson AFB, OH, March 2020.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704–0188

1. REPORT DATE *(DD–MM–YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From — To)*
26-03-2020	Master's Thesis	SEP 2018 - MAR 2020

4. TITLE AND SUBTITLE

Meta Learning Recommendation System for Classification

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Williams, Clarence, O. 1st Lt, U.S. Air Force

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 Hobson Way
WPAFB OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT-ENS-MS-20-M-181

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Intentionally Left Blank

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

DISTRIBUTION STATEMENT A:
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

13. SUPPLEMENTARY NOTES

This work is declared a work of the U.S. Government and is not subject to copyright protection in the United States

14. ABSTRACT

A data driven approach is an emerging paradigm for the handling of analytic problems. In this paradigm the mantra is to let the data speak freely. However, when using machine learning algorithms, the data does not naturally reveal the best or even a good approach for algorithm choice. One method to let the algorithm reveal itself is through the use of Meta Learning, which uses the features of a dataset to determine a useful model to represent the entire dataset. This research proposes an improvement on the meta-model recommendation system by adding classification problems to the candidate problem space with appropriate evaluation metrics for these additional problems. This research predicts the relative performance of six machine learning algorithms using support vector regression with a radial basis function as the meta learner. Six sets of data of various complexity are explored using this recommendation system and at its best, the system recommends the best algorithm 67% of the time and a "good" algorithm from 67% to 100% of the time depending on how "good" is defined.

15. SUBJECT TERMS

Meta Learning, Meta Modeling, Recommendation System, Algorithm Selection

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Dr. Jeffery D. Weir, Ph.D., AFIT/ENS
U	U	U	UU	105	19b. TELEPHONE NUMBER *(include area code)* (937) 255-3636, x4523; jeffery.weir@afit.edu