Journal of Machine Learning Research 1 (2015) 1-48 Submitted n/a; Published n/a

A General Tool for Learning-Algorithm Optimization and Comparison

Mark Hamilton MARK.HAMILTON@YALE.EDU
YALE UNIVERSITY
NEW HAVEN, CT

Editor:

Abstract

I have utilized the scikit-learn (sklearn) and pybrain python libraries to create a modular learning-algorithm comparison software. This software consists of my own python module and fitting programs for pre-processing and regressing data of arbitrary size and dimension. I have also created a sklearn-integrated neural-network regression class using the machinery from pybrain. This addition was needed due to sklearn's lack of a general neural-network regression algorithm that supports deep networks. Furthermore, I created an optimization program that performs k-fold cross validation over a regression pipeline that includes imputing, scaling, dimensionality reduction, and fitting with multiple regression algorithms. This program is modular and easily supports the addition of more fitting algorithms and optimization parameter spaces. Currently, this program optimizes over kernel ridge, support vector, random forest, gradient boosting, and neural network regression algorithms. I have applied this software to astronomical data to calculate photometric redshifts for Active Galactic Nuclei (AGN) with only 250 data points. Due to high variability and lack of research, these objects pose one of the greatest challenges in the photometric redshift fitting community. My software yields improved fitting accuracy when compared to other photometric-redshift algorithms such as Lephare.

1. Motivation

When confronted with a challenging data analysis problem it is often difficult to know where to begin, especially if little is known about the structure and nature of the data. If the dataset under investigation is small, obtaining an effective model of the data can be a significant challenge. Thankfully, many general regression algorithms have been created to fit data from arbitrary function classes. However, many of these algorithms have a large and non-convex meta-parameter space that is not well understood. Through the course of this project, I have created software that is general enough to handle many different types of regression problems through the lens of model search and selection. This software is designed to be used as an off the shelf model selector that can work on arbitrary datasets.

2. Astrophysics Background

For many decades, it has been known that our understanding of physics is woefully incomplete. Scientists have observed massive effects called "Dark energy" and "Dark matter" which defy all standard explanations. Consequently, we have created several new theories

that offer tantalizing unifications and explanations, but have not been validated by any experimental evidence. Today, cosmologists are interested in studying the universe at the largest of scales to glean clues about possible new models of our universe. One of the many frontiers currently under research involves the distribution of galaxies as a function of space and time. By better understanding this distribution, we can get better estimates of our universes fundamental parameters and can uncover clues from a time when the structure of our universe was very different from that of today.

Recent observations have showed that almost all galaxies contain a large super-massive black hole at their center, which holds stars in a galactic orbit. At certain stages of a galaxy's life, a large portion of galactic material will spiral into the black hole creating a significant amount of friction, heat, and light. We observe these high-energy objects as incredibly intense points of light called quasars, blazars, or more generally, Active Galactic Nuclei (AGN). Recently, researchers have discovered that many of these objects might be obscured by large toroidal dust clouds of debris that block the bright light emanating from the AGN.[2] Consequently, this has sparked a large movement to measure the true distribution of the AGN in space and time, which requires knowing their distance from earth. This is calculated by finding a bright peak in the spectrum of a galaxy called an emission line and determining its redshift. Physically speaking, these emission lines come from the quantized emission of light from electron transitions between orbital states in atoms such as helium. A redshift refers to how much the light waves have been stretched and elongated by the expanding universe. Larger distances from earth entail longer light travel times, which implies more stretching by the expanding universe. Emission lines occur at very specific wavelengths, so observing the magnitude of the shift can give incredibly accurate measurements of galaxy distance.

For years, researchers have been patiently collecting data from several observatories and aggregating the results in large databases. However, taking full spectrum images of all of the objects observed is costly, time consuming, and unfeasible on a large scale. Consequently, most datasets only have spectra for less than 10% of the observations.[6] The remainder of the observations contain only a few (3-15) light intensity measurements at specific wavelength bands. Because of this limitation, conventional redshift calculations that rely on a well-resolved characteristic peak fail, and approximation methods must be used. This problem spawned a new technique called "photometric redshift" (photz) fitting where redshifts are estimated from the few intensity bands available. Conventional approaches involve fitting a template galactic spectrum to each galaxy's light intensity measurements. This template usually depends on a few parameters such as scale, redshift, and type of galaxy. The parameters are then tuned using maximum likelihood estimation, and the corresponding redshift estimate is returned. These methods perform fairly well on normal galaxies with several (≈ 30) wavelength bands. However, these methods are inaccurate and imprecise when applied to AGN, and require specific templates and highly tuned fitting routines.[1] This past summer, I searched the parameter space for weeks to find a fitting routine that could accurately predict redshifts for these types of galaxies. However, the search was unsuccessful as I could not improve beyond a slight correlation between predicted and the actual or "spectroscopic redshift" (specz) obtained from the full spectra, as can be seen in Figure 2. Furthermore, these objects are rare and need to be observed by satellite x-ray observatories, I was limited to a dataset of size ≈ 250 objects, many of which

had missing entries. This motivated me to pursue an analysis method that could effectively handle small sample sizes, noisy data, and missing values.

3. Data

The data used in this project has been compiled from a list of x-ray selected AGN from the Chandra, XMM, and Stripe-82 x-ray observatories. The data is then cross-matched using a maximum likelihood estimator based on position and brightness to objects in the Sloan Digital Sky Survey (SDSS), GALEX, UKIDSS, and WISE astronomical surveys. This results in a list of objects selected by x-ray observation that have magnitude (brightness) measurements in one or more of the astronomical surveys. Each survey has its own set of wavelength bands for a maximum total of 15 magnitude measurements. These magnitudes are acquired by taking the image from the telescope, and "extracting" the brightness by integrating a small circular region around the galaxy. There are several different competing ways to perform this extraction, each offering advantages in different parameter regimes. It was not clear a priori which magnitude type would yield accurate results in a fitting program. As a result, a brute force check on all magnitude type combinations was performed with cross validation to determine the most effective combination for the Lephare redshift fitting algorithm. Once the proper magnitude types have been selected and queried, the data are converted to the same unit scale (AB magnitudes) and several transformations are applied from the Schlegel dust maps to correct for galactic dust reddening from the milky-way.[4] This ensures that Lephare can properly fit the galaxy templates to the data without having to add a dust correction parameter, which adds unnecessary degrees of freedom to the problem. Furthermore, AGN magnitude can vary drastically on timescales as small as months, and each survey (SDSS, Galex etc) has collected data at different times of the year.[5] To correct for this variability, the data-mining method pulls the observation that is temporally closest to the SDSS measurement. Throughout the course of this work I used only the objects with observations in all four surveys.

4. Fitting Process

On the large scale, the fitting process contains several modular units that each perform a separate computationally intensive task. This modularity minimizes the need to re-compute optimized parameters. The first unit performs a grid search optimization over each algorithm under investigation. The results of this optimization are "pickled" (Python's version of saving any object to a file) into a file and then loaded by the two result plotters. The first performs a bootstrapped plot using the best parameter results from the optimization. This bootstrapping helps to characterize the variability of the algorithm despite a lack of data. The second feeds the best results into a bagging regressor with 30 objects to decrease variability. This computation is usually lengthy, so the results of this are pickled into a file for re-plotting at any time.

The fitting process has been implemented almost entirely in the sklearn and pybrain packages of python. The first step involves reading a parameter file called $''ml\_param.py''$ which includes the names of various input and output files and several metaparameters. This keeps the codes short and is intended to be as general as possible for scripting and

chaining these codes together. The second step in the process involves parsing the data file and extracting the relevant columns, which are stored in the $''features.txt''$ file. This allows for easy updating of the input data and features used in the fitting process. All three main codes then standardize the data by replacing various missing value placeholders to -99. Next, the data is converted to numpy arrays for processing with sklearn.

The code is based around the sklearn pipeline object, which is a general method for chaining together preprocessing, dimensionality reduction, and regression procedures into a single object which can be manipulated. The use of this pipe object adds conceptual simplicity, and allows for grid searches over the parameters of the entire process.

The first object in the pipeline is an imputer, which replaces the $''-99''$ values with the mean of the column that the data is taken from. This improves fitting performance by mitigating the effect of missing data in a somewhat neutral manner. The second object in the pipeline is a scaler, which subtracts the mean of the data and scales it to have a standard deviation of 1. This procedure helps standardize the data fed into the algorithms, and has been shown to greatly improve performance in fitting. Furthermore, this step is crucial for the dimensionality reduction phase that is usually scale dependent. The next phase of the pipeline is the dimensionality reduction step. This takes the form of a principle component analysis with optional whitening. The code is designed so that this can be easily switched out for kernel PCA, ICA or other dimensionality reduction procedures. Further iterations of the code will all of these algorithms in the optimization procedure. This step is crucial in high dimensional problems because of the "curse of dimensionality" where higher dimensional datasets result in less training data per unit volume, which often decreases regression performance. [3] The final stage of the pipeline is the regressor object that can be trained and used to predict the response variable.

In the optimization code, the best set of meta-parameters is searched for through several grid searches on the analysis pipeline. More specifically, the code loads a list of regressors and optimization parameters from the file "$ml\_classifier\_list.py$", which keeps the code modular and cuts down on clutter in the main code. It also facilitates the addition of more algorithms and parameter spaces to search over. A joint dictionary is then constructed which adds the PCA and regressor optimization parameters for simultaneous optimization. This dictionary is the object that sklearn's grid search object scans over. The grid search performs a 10-fold cross validation for each set of parameters and automatically determines the best set of metaparameters. This grid search is then applied to each algorithm in the list, resulting in a list of grid search regression objects. This list of grid search regressors is then pickled for later use and plotting. Furthermore, the code prints the results of every grid search combination to a user specified text file that is by default called "$optimization\_verbose\_n.txt$". It also prints the best results to "$optimization\_best\_n.txt$" where n is an integer representing the number of times the code has been run. The optimization is also general enough to allow for user created metrics. This analysis uses the normalized median absolute deviation (σ_{NMAD}) , which is more resistant to outliers than the MSE. [1]

The code currently optimizes over five types of algorithms. Namely, Kernel Ridge regression (KR), Support Vector Regression (SVR), Random Forest Regression (RF), Gradient Boosting Regression (GBR), and a multi-layered convolutional neural-network (NN). These algorithms were chosen because they are some of the most popular and general methods

to fit arbitrary datasets. The parameters for the KR and SVR are optimized over several different kernels, regularization, and penalty parameters. The optimizations for RF, and GBR run over the number of estimators and the maximum depth of the trees. Additionally, GBR runs over different types of loss metrics. The NN optimization runs over the network structure which defines the number of layers and the number of nodes at each layer and the number of training iterations. These allow for the metaparameter space of each algorithm to be explored thoroughly. Sklearn takes in these parameters as lists of dictionaries and a full breakdown of the parameters used can be found in Appendix C.

In the result-plotting and ensemble-plotting portions of the code, the results of this optimization are loaded from the pickled file and retrained on the data. The ensemble-plotting code creates a list of new regressors consisting of the best parameters of each algorithm fed through a bagging regressor. This bagging regressor functions by creating n copies of the original regressor, and training the copies on several different subsets of the training data. The final results are then averaged to decrease the variance. This has been shown to improve performance of high variance algorithms such as neural networks and decision trees. The code then uses bootstrapping on these bagging regressors to calculate trial vs test errors on random selections of the data. It then plots of the spectroscopic and photometric redshifts for all of these random selections on the same graph. This bootstrapping procedure helps maximize the amount of training and testing data that can be used and visualized without jeopardizing the information barrier between training and test sets. Like the grid search, this calculation is usually lengthy, so it has options for running on multiple cores and pickles its results to an output file. If the code detects the presence of this output file, it will skip the bootstrapping calculation and just plot the results. This adds flexibility for larger scale datasets and larger numbers of objects in a single bag. The code also has options for sub-sampling and reducing the number of folds in the cross validation for larger datasets.

4.1 Neural Network Addition to Sklearn

Using the python library Pybrain, I created a neural network regressor class which integrates with sklearn's pipeline objects and metrics. This class consists of several hyper-parameters including the number of training iterations, the network structure, and the response function for the hidden layers. The network structure is a list of positive integers whose length will determine the number of layers of the network, and whose values will determine the number of neurons in the respective layer. These layers are connected with full connections to the adjacent layers with a response type indicated by the corresponding hyper-parameter. The network uses a linear input layer with number of nodes equal to the number of inputs and a single linear output node. The default is the sigmoid function, but the class also supports softmax, linear, hyperbolic tangent, and Gaussian response functions. The class's default scorer is the MSE but this can be overridden by supplying a new scoring function to the pipeline.

5. Results

The results obtained show significant improvement from the default settings to the optimized settings. Furthermore, in three of the five algorithm categories, bagging significantly reduced

Algorithm	PCA comps	Whitening	Kernel, Response, or Loss	Gamma	Alpha/C	Iterations or Number of objects
KR	15	T	Deg 3 poly with offset	0	1	n/a
SVR	15	F	RBF	.001	100	n/a
RF	10	T	MSE	n/a	n/a	100
GBR	15	T	MSE	n/a	n/a	50
NN	15	F	Sigmoid	n/a	n/a	800

Figure 1: Summary of optimization results. "Kernel" applies to KR and SVR, "Loss" applies to RF and GBR, and "Response" applies to NN. "Alpha" applies KR and "C" applies to SVR. "Iterations" applies to NN and "Number of objects" applies to RF and GBR.

variance and improved fitting performance with respect to percentage of outliers (η) and the normalized median σ_{NMAD}. Bagging did not improve RF and GBR, perhaps because these algorithms already employ a bagging strategy. All methods made significant improvement over the Lephare Parametric fit. Optimized parameters are shown in Figure 1 and regression results at each stage of the procedure are shown in Figures 3,4,5 .

6. Conclusions

Completely out of the box, this code has made significant improvement on photometric redshift fitting when compared to standard parametric techniques such as Lephare. The code took approximately two days to run and converged to very similar solutions during subsequent re-runs. Furthermore, this code was able to make these significant improvements with an exceedingly small dataset. With a larger dataset, these methods will undoubtedly become more effective. This code performs a significant amount of analysis in a convenient and modular format for easy application to other regression problems.

7. Future Work

While this code is already fairly general, it could be improved by the addition of more algorithms and a more general procedure for constructing and optimizing over multiple algorithms. Furthermore, the addition of cyclic coordinate optimization of metaparameters would be a very nice addition to discover more optimal sets of metaparameters that can scale to larger problems where dictionary grid searches might be computationally infeasible. To improve computational performance, I plan to re-develop the neural-network object using Theano and pylearn2. These systems have a steeper learning curve, but offer GPU processing and can run approximately 100x faster than pybrain. Considering that the Neural Network algorithm is the majority of the running time, this is a much needed improvement for future scalability. Furthermore, I would like to implement a symbolic regressor object and create an intuitive environment to evolve a family of good symbolic estimators for an arbitrary dataset. I plan to make this a separate part of the code that can be run alongside

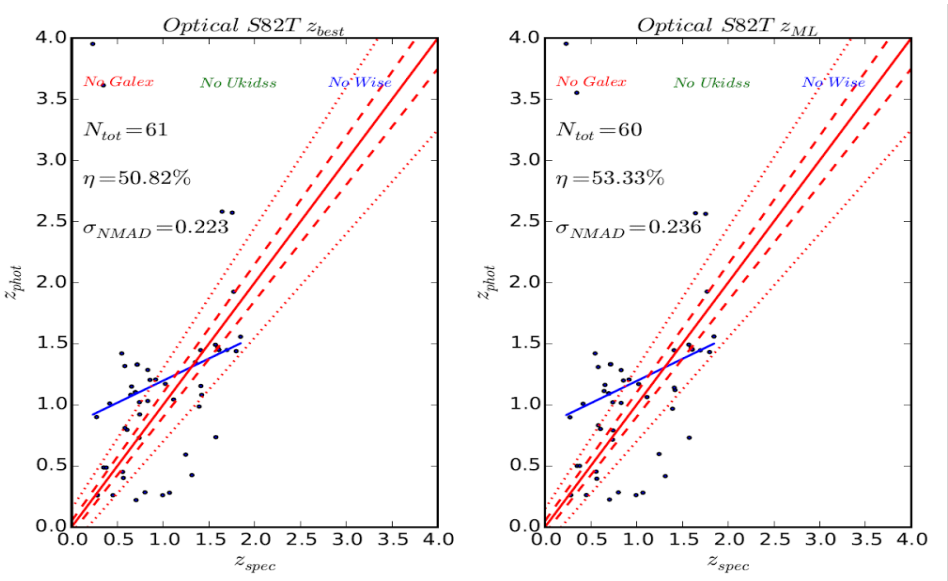

Figure 2: Original Lephare fit on the testing set comparing the spectroscopic redshift and the photometric redshift. z_{best} corresponds to the best lephare output and z_{ml} corresponds to the maximum likelihood estimation. Blue lines represent linear regressions through the points.The solid line is the $z_{phot} = z_{spec}$ relation. The dashed lines are $z_{phot} = 0.05(1+z_{spec})$. The dotted lines are $z_{phot} = 0.15(1+z_{spec})$. Sources that lie outside the dotted lines are defined as outliers. Lephare was run with the templates and analysis flow used in [1].

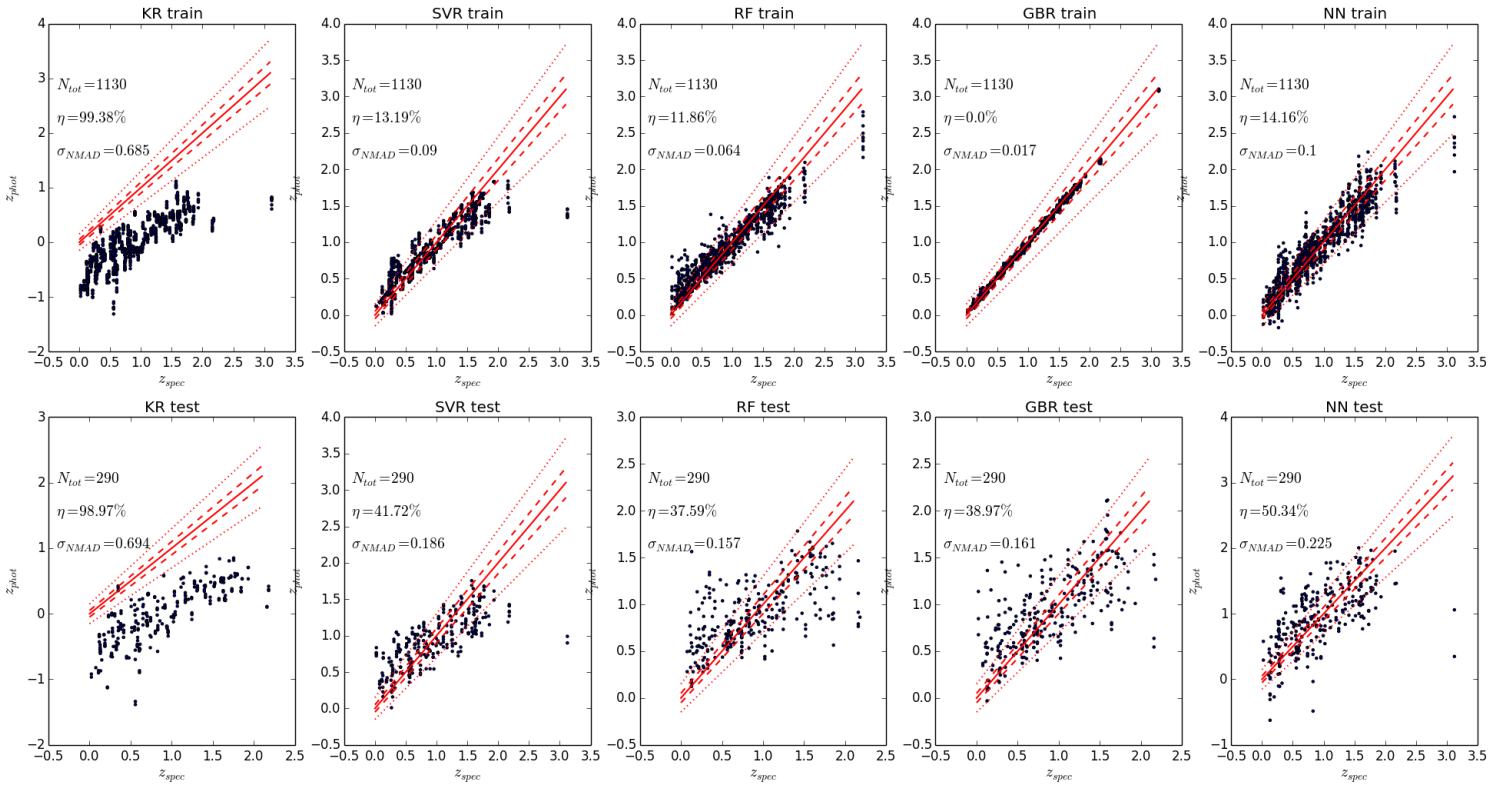

Figure 3: Photometric redshift estimate comparison between different algorithms with default settings. Algorithms from left to right include Kernel Ridge (KR), Support Vector (SVR) , Random Forest (RF) , Gradient Boosted (GBR) , and a Convolutional Neural Network (NN). Training sets are shown above testing sets and the red lines are the same from figure 2

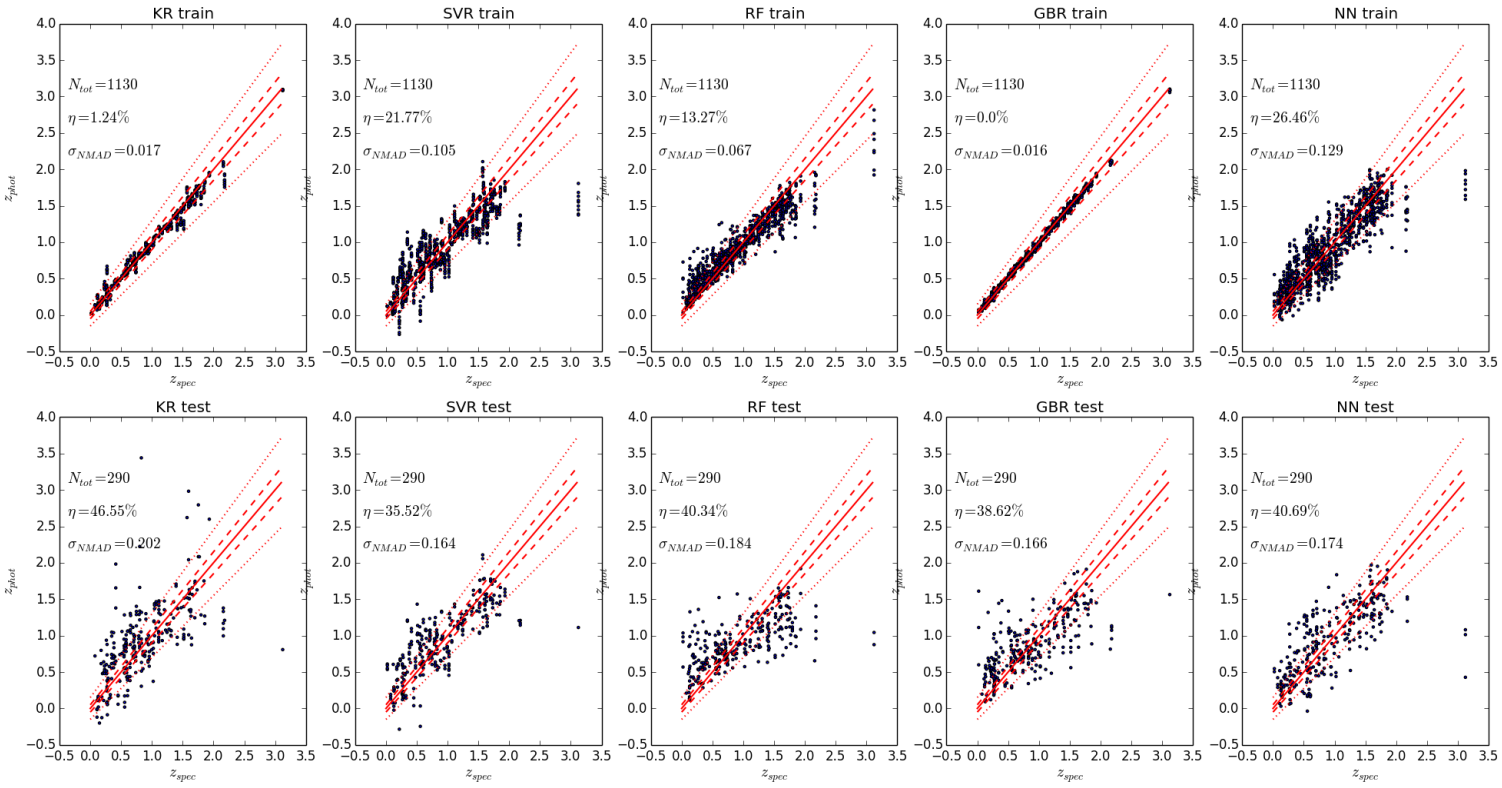

Figure 4: Photometric redshift estimate comparison between different algorithms after optimization. Algorithms from left to right include Kernel Ridge (KR), Support Vector (SVR) , Random Forest (RF) , Gradient Boosted (GBR) , and a Convolutional Neural Network (NN). Training sets are shown above testing sets and the red lines are the same from figure 2

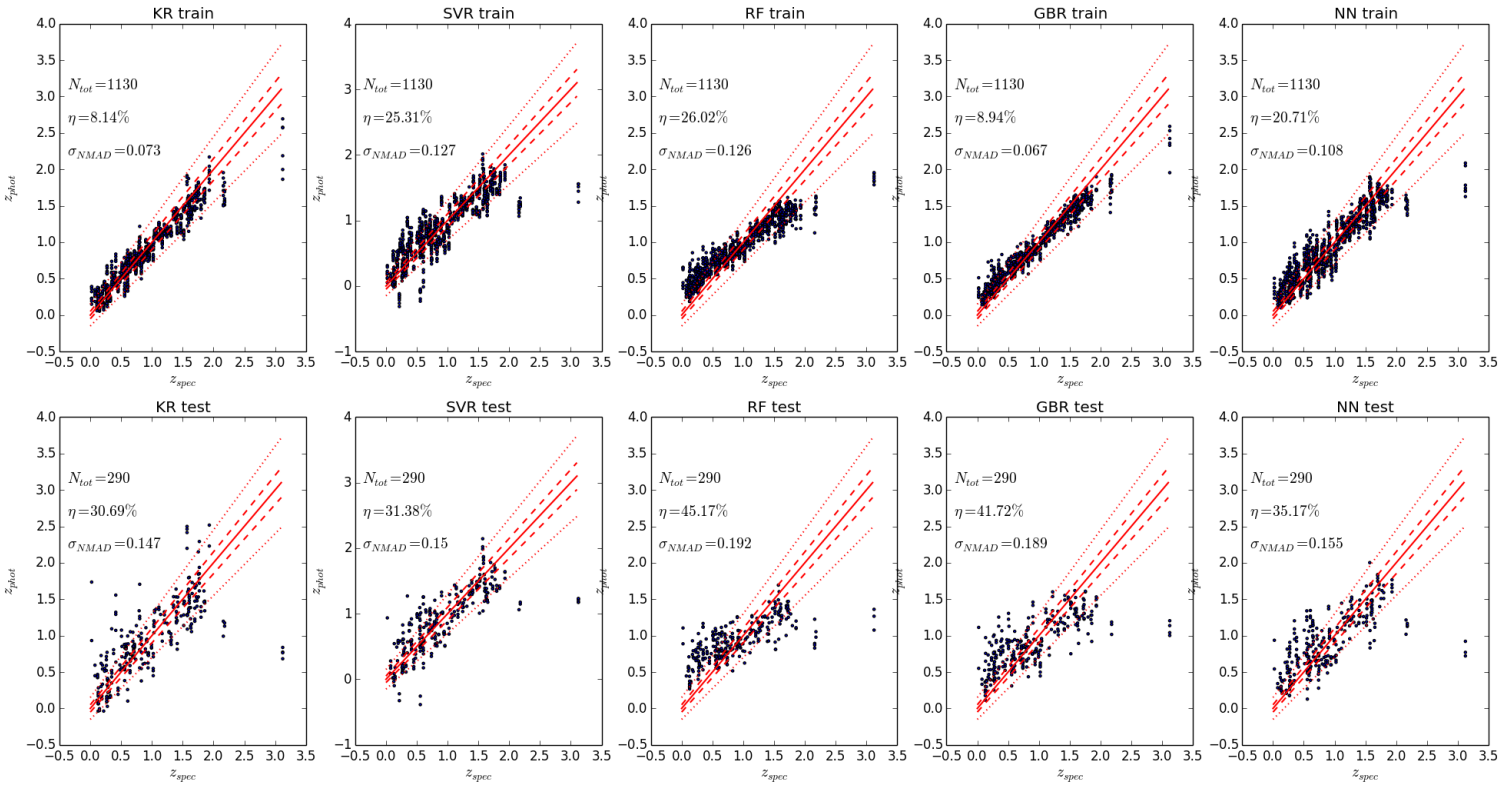

Figure 5: Photometric redshift estimate comparison between different algorithms after optimization and 30-object bagging. Algorithms from left to right include Kernel Ridge (KR), Support Vector (SVR) , Random Forest (RF) , Gradient Boosted (GBR) , and a Convolutional Neural Network (NN). Training sets are shown above testing sets and the red lines are the same from figure 2

the first optimization. With this addition, the code would have another set of tools for unraveling the internal structure of the data.

Acknowledgments

Thanks to Professor Meg Urry and Post Doctoral student Stephanie LaMassa for supervising the project and offering an immense amount of help and guidance throughout the process. Thanks also to Professor Sahand Negahbahn for his many informative discussions about machine learning methods. Thank you Dr. Mara Salvato for your many hours of lephare troubleshooting and helpful tips.

References

[1] Fotopoulou, S., Salvato, M., Hasinger, G., Rovilos, E., Brusa, M., Egami, E., Lutz, D., Burwitz, V., Henry, J., Huang, J., et al. Photometry and photometric redshift catalogs for the lockman hole deep field. *The Astrophysical Journal Supplement Series 198*, 1 (2012), 1.

[2] Guainazzi, M., Matt, G., and Perola, G. C. X-ray obscuration and obscured agn in the local universe. *Astronomy & Astrophysics 444*, 1 (2005), 119–132.

[3] Hastie, T., Tibshirani, R., Friedman, J., and Franklin, J. The elements of statistical learning: data mining, inference and prediction. *The Mathematical Intelligencer 27*, 2 (2005), 83–85.

[4] Schlegel, D. J., Finkbeiner, D. P., and Davis, M. Maps of dust infrared emission for use in estimation of reddening and cosmic microwave background radiation foregrounds. *The Astrophysical Journal 500*, 2 (1998), 525.

[5] Uttley, P., and McHardy, I. M. A brief review of long-term x-ray and optical variability in radio-quiet agn. *Progress of Theoretical Physics Supplement 155* (2004), 170–177.

[6] York, D. G., Adelman, J., Anderson Jr, J. E., Anderson, S. F., Annis, J., Bahcall, N. A., Bakken, J., Barkhouser, R., Bastian, S., Berman, E., et al. The sloan digital sky survey: Technical summary. *The Astronomical Journal 120*, 3 (2000), 1579.

Appendix A: Algorithm Optimization code.

All code has been moved to:
`https://github.com/mhamilton723/model_selection`

Appendix B: Main code module

All code descriptions have moved to:
`https://github.com/mhamilton723/model_selection`

Appendix C: Regressor Optimization Parameter File

All code descriptions have moved to:

All code descriptions have moved to:

`https://github.com/mhamilton723/model_selection`