Online model stacking

Jeremy Coyle
Sam Lendle
Sara Moore

May 15, 2013

1 Introduction

The quickly evolving field of large-scale machine learning, or "data mining," brings with it several challenges. First, interesting large datasets, while perhaps not a problem to store, may be too large to hold in memory in their entirety while processing. A related issue is that of a dataset that is never complete; observations, or groups of observations, are constantly being added to "streaming" data. In both cases, the ability to fit models piecemeal and later combine estimates would be advantageous.

An "online" algorithm does just that. "Online" learning is a paradigm in which models are fit to data coming in a "stream," eliminating the need to store all past observations or fit on the entire dataset en masse. An online algorithm, in the form we will discuss below, takes each block of incoming data and uses it to update the fit of the model. The data are subsequently discarded. However, before the fit is updated, each new block can also be used to assess the performance of the model (a sort of cross-validation). Any algorithms that can be fit with stochastic gradient descent are a perfect fit for this situation.

Another problem encountered in data mining is the selection of the best learning algorithm. A wide variety of prediction algorithms are available, but for any new prediction problem, it's not clear which one will best capture the true relationship between the features and the outcome.

One potential solution is the use of "model stacking," an ensemble method where many algorithms, possibly from different types of models, are combined to form a final estimator. This is accomplished by training the individual algorithms concurrently, then combining them via a weighted average, using using either hold-out data or cross-validation to avoid overfitting. One example of such an ensemble method is Super Learner, implemented in `R`, which combines competing algorithms by minimizing a suitable loss function of a linear combination of their predictions under cross-validation. This combined approach has been shown to perform asymptotically as well as or better than the "best" algorithm in the library of algorithms (van der Laan et al., 2007).

In this project, we seek to find a combined solution to the two problems detailed above. Specifically, we aim to develop an online/streaming approach to training stacking algorithms, and to do so efficiently by training each algorithm on each data block concurrently.

2 Datasets

As the primary focus of this project was evaluating a method, we allowed the method to dictate the data we used. Also, to test the algorithm under a variety of conditions, we chose to evaluate the algorithm on multiple sets of data.

The first of these datasets was the most recent "dump" from Wikipedia. We took a similar approach to that in assignment 3, using single words from each article's text to predict (dichotomous) inclusion in a parent category of "Mathematics." This dataset met our qualifications of being large enough to warrant exploration of an online approach to its analysis; our training set was composed of approximately $4,500,000$

observations, while the validation was run on 100,000 articles not included in the training set. For the feature space, a dictionary was formed from the 200,000 most common words in the data, excluding stopwords.

The second dataset that was used to evaluate the algorithm was also a "dump," this time of Stack Overflow questions through July 31, 2012 (downloaded from http://www.kaggle.com). This data was advantageous, as it was not only large, but could theoretically fit into a streaming context. We again formed a binary outcome for prediction, which was, for each question, whether or not its status was eventually "closed." The training set was approximately 3,000,000 questions, with a 10% validation set of approximately 300,000 questions. The features used for prediction included words in the question's title and body (not including those longer than 15 characters), question tags, posting user reputation and the number of undeleted questions by that user. Instead of pre-parsing the question text to form a dictionary, and because we were not concerned with *which* words were most predictive of our outcome of interest, the text was instead hashed for efficiency.

3 Analysis methods

In this section we describe a model stacking algorithm that can be trained in a sequential way on blocks of data based on the SuperLearner algorithm (van der Laan et al., 2007). We focus on our implementation choices and also describe possible modifications or generalizations to our approach.

3.1 Preliminaries and notation

Let $X = (Y, W)$ denote a random variable where $Y \in \mathbb{R}$ and $W \in \mathbb{R}^p$. Let $X_1, X_2, \ldots$ denote a sequence of independent and identically copies of X with distribution P_0. In general, we are interested in estimating the function

$$Q_0(W) = \arg\min_{Q \in \mathcal{Q}} E_0 L(X, Q)$$

where L is some loss function, E_0 denotes expectation with respect to the distribution P_0, and $\mathcal{Q}$ is some set of possible functions Q, possibly unrestricted. Possible choices for L are the squared error loss and the negative log likelihood loss. For the squared error loss, $(Y - Q(W))^2$, Q_0 corresponds to the conditional mean $E_0(Y \mid W)$ (assuming $\mathcal{Q}$ is rich enough). If L is the negative log likelihood loss function, then Q_0 is the conditional probability density function of the distribution P_0, $p_0(Y \mid W)$, assuming $p_0 \in \mathcal{Q}$. For example, Y is Bernoulli then the negative log likelihood loss is $-[Y \log(Q(W)) + (1-Y)\log(1-Q(W)]$, and Q_0 corresponds to $p_0(Y = 1 \mid W)$. In this case, $p_0(Y = 1 \mid W)$ is equivalent to $E_0(Y \mid W)$, so either loss function can be used for binary Y.

Additionally, let an overbar denote a block of observations arranged in a matrix. Thus $\bar{Y} = (Y_m, Y_{m+1}, \ldots Y_{m+b-1})'$ is a $b \times 1$ matrix of outcomes Y, and let $\bar{W} = (W_m, W_{m+1}, \ldots W_{m+b-1})'$ is a $b \times p$ matrix of features, where the ith, jth corresponds to the jth feature of observation $m + i - 1$. Let $\hat{Q}$ denote an estimate of Q_0.

3.2 Main Algorithm

The algorithm we implement is intended for a single pass over a very large dataset, or for applications where data is essentially infinite, coming in a stream which may not be stored permanently.

Details of the algorithm are described in algorithm 1. Functions INITIALIZEBASE, UPDATEBASE, PREDICTBASE depend on the sort of base learning algorithms that are included are described in section 3.3. The weighted average of base learning algorithms is based on a moving window of the last w observations. We keep a $w \times m$ matrix Z of predictions of the last w observations for each estimator $\hat{Q}_i$. We also keep the last w outcomes Y in memory, called $\bar{Y}'$. The prediction for each block of observation is from estimators that have not yet been trained on that block to avoid overfitting. As described, we assume $w \geq b$ but the algorithm can easily be modified for the case $w < b$, which is simpler because Z and $\bar{Y}'$ do not need to be saved between steps.

The weighted average of base learners is defined as the minimizer of $||\bar{Y}' - Z\alpha||$ subject to $\alpha_i > 0$ for all i, normalized to sum to one. Other loss functions can be used to combine the base algorithms.

After INITIALIZE is run, the algorithm is trained sequentially on blocks of data by calling UPDATE which gets a new block of data and updates each base algorithm and the weighted SuperLearner combination. At any time, predictions can be made on one or more new values of W with a call to PREDICT. Although not described in algorithm 1, in our implementation we estimate the risk for each base model and for the SuperLearner combination before updating any models on the new block. In this way, we get a valid estimate of the out-of-sample risk almost for free computationally, because we need predictions on the new block before training the base algorithms anyway.

3.3 Base learning algorithms

A base learner can be any learning algorithm that can be trained in an incremental way, such as with stochastic or mini-batch gradient descent (Bottou, 2010) or incremental L-BFGS (Langford, 2013; Schraudolph et al., 2007). Such an algorithm must implement an UPDATEBASE method, taking a new block of Ys and Ws, and updating the estimate accordingly, and a PREDICTBASE method, which returns predictions for a matrix of (new) W values. The INITIALIZEBASE method should initialize the estimator in a reasonable way before it sees the first block of data.

We implemented an intercept only model ("MeanLearner"), logistic regression without regularization and with ℓ_1 or ℓ_2 ("LogisticRegression", "LassoLogisticRegression", and "RidgeLogisticRegression", respectively) and a linear SVM using mini-batch gradient descent with AdaGrad scaling (Duchi et al., 2010). To calculate predicted probabilities from SVM, we wrapped the linear predictor in an additional logistic regression as proposed by Niculescu-Mizil and Caruana (2005).

3.4 Possible modifications

The algorithm we describe in section 3.2 can be modified in various ways to improve performance. A simple modification would be to allow each base algorithm to use different block sizes, where block size could be a tuning parameter for each base algorithm.

Base algorithms could be added or removed from the library of algorithms adaptively. For example, if for some number of blocks, some algorithms have weight 0 in the weighted combination of algorithms, we could stop training them, thereby speeding up training time. Additionally, if we have a range of regularization parameters λ for LassoLogisticRegression, say, and the largest λ consistently has the highest weight in the combination of algorithms, we could add an additional LassoLogisticRegression with a larger λ to search for a better regularization.

Algorithm 1 avoids overfitting by only using predictions from new blocks of data before training on them. This works if we only make one pass over the data, but will fail if we see the same data more than once. To allow for multiple passes over the data, instead of updating the weighted combination based on all observations, we could hold out a small random subset of observations that are never used for updating the base algorithms. We could take this a step further and implement full cross-validation, training multiple copies of each algorithm with different validation sets held out.

3.5 Performance Evaluation

In addition to estimating the risk during training, we also evaluate the performance of each base algorithm and the SuperLearner combination on an independent test set. The performance was assessed by MSE, prediction accuracy, area under the receiver operating characteristic curve (AUC) and F1.

Algorithm 1 Online Model Stacker

function INITIALIZE(blockSize, windowSize, listOfBaseLearners)
  $b \leftarrow blockSize$
  $m \leftarrow$ LENGTH(listOfBaseLearners)
  $w \leftarrow windowSize$
  $s \leftarrow 0$ ▷ s is step number
  for $i = 1 \rightarrow m$ **do**
    $\hat{Q}_i^0 =$ INITIALIZEBASE($listOfBaseLearners[i]$)
  end for
  $\bar{Y}' \leftarrow (w \times 1)$ matrix with all elements equal to 0
  $Z \leftarrow (w \times m)$ matrix with all elements equal to 0
  $\hat{\alpha} \leftarrow (m \times 1)$ matrix with all elements equal to $1/m$
end function
function GETNEXTBLOCK
  $\bar{Y} \leftarrow$ next $b \times 1$ block of Ys
  $\bar{W} \leftarrow$ next $b \times p$ block of Ws
  return $(\bar{Y}, \bar{W})$
end function
function UPDATE
  $s \leftarrow s + 1$
  $(\bar{Y}, \bar{W}) \leftarrow$ GETNEXTBLOCK
  $\bar{Y}'[1 : (w - b), 1] \leftarrow \bar{Y}'[(b + 1) : w, 1]$
  $\bar{Y}'[(w - b + 1) : w, 1] \leftarrow \bar{Y}$
  if $s > 1$ **then**
    $Z[1 : (w - b), 1 : m] \leftarrow Z[(b + 1) : w, 1 : m]$
    for $i = 1 \rightarrow m$ **do**
      $Z[(w - b + 1) : w, i] \leftarrow$ PREDICTBASE($\hat{Q}_i^{s-1}$, $\bar{W}$)
    end for
    $\hat{\alpha} \leftarrow \underset{\alpha:\alpha_j>0 \forall j \in \{1,\ldots,m\}}{\arg\min} ||\bar{Y}' - Z\alpha||$
    $\hat{\alpha} \leftarrow \hat{\alpha}/||\hat{\alpha}||_1$
  end if
  for $i = 1 \rightarrow m$ **do**
    $\hat{Q}_i^s \leftarrow$ UPDATEBASE($\hat{Q}_i^{s-1}$, $\bar{W}$)
  end for
end function
function PREDICT($\bar{W}'$)
  $b' \leftarrow$ NUMROWS($\bar{W}'$)
  for $i \in 1 \rightarrow m$ **do**
    $Z'[1 : b', i] \leftarrow$ PREDICTBASE($\hat{Q}_i^s$, $\bar{W}'$)
  end for
  return $Z'\hat{\alpha}$
end function

4 Results

Initial testing of the online Super Learner algorithm was conducted using the Wikipedia dataset. First, we investigated using Super Learner to combine the same base learning algorithm (logistic regression) with different learning rate parameter values. The algorithm selected a linear combination of those values that performed well, selecting learning rates around 0.1 (fig. 1). The algorithm tended to prefer learners with faster rates more initially, but tended to preferring slower learning rates by the end of the training data. In terms of the risk estimated on the training set, Super Learner performed better than any of the base learners (fig. 2). The algorithm also performed well on the validation set (fig. 3).

Similarly, Super Learner performed well when given a collection of the same base learning algorithm with different tuning parameters, here the regularization parameter λ for logistic regression with lasso (ℓ_1) regularization (figs. 4 and 5). For the wikipedia dataset, the algorithm tended to prefer models with minimal regularization.

Next, we assessed the performance of Super Learner on a combination of different model types. The algorithm put most weight on lasso and logistic regression, learners that perform well individually (figs. 6 and 7). Again, Super Learner did better than any of the base learners on both the training and testing sets (fig. 8).

Finally, using the Stack Overflow dataset, we assessed the performance of Super Learner supplied with a very large number of base learners. Learners with a range of different learning rates and regularization parameters were used. A total of 96 learners was used. After iterating over the full training set, the algorithm put nonzero weight on only six algorithms, a mix of logistic regression both unregularized and with ℓ_1 and ℓ_2 penalties (fig. 9). Assessing performance on the validation set, Super Learner found a linear combination of learners that outperformed any individual learner (fig. 10).

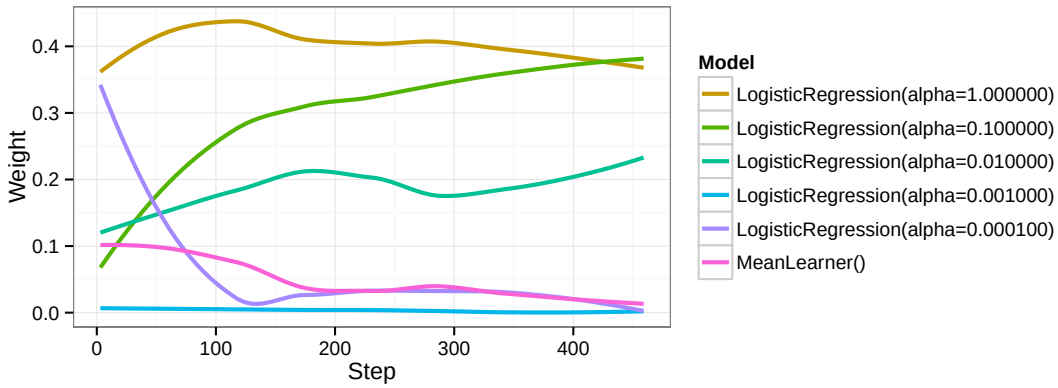

Figure 1: Weights for a collection of LogisticRegression learners with different learning rates (Wikipedia)

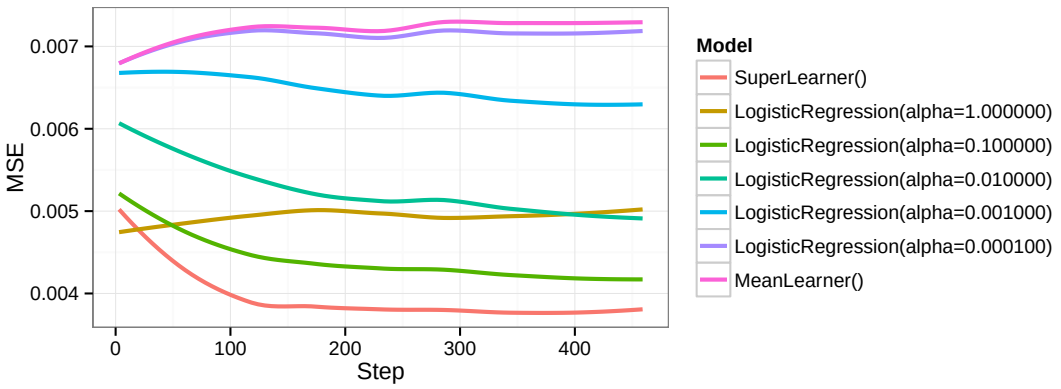

Figure 2: MSE for a collection of LogisticRegression learners with different learning rates (Wikipedia)

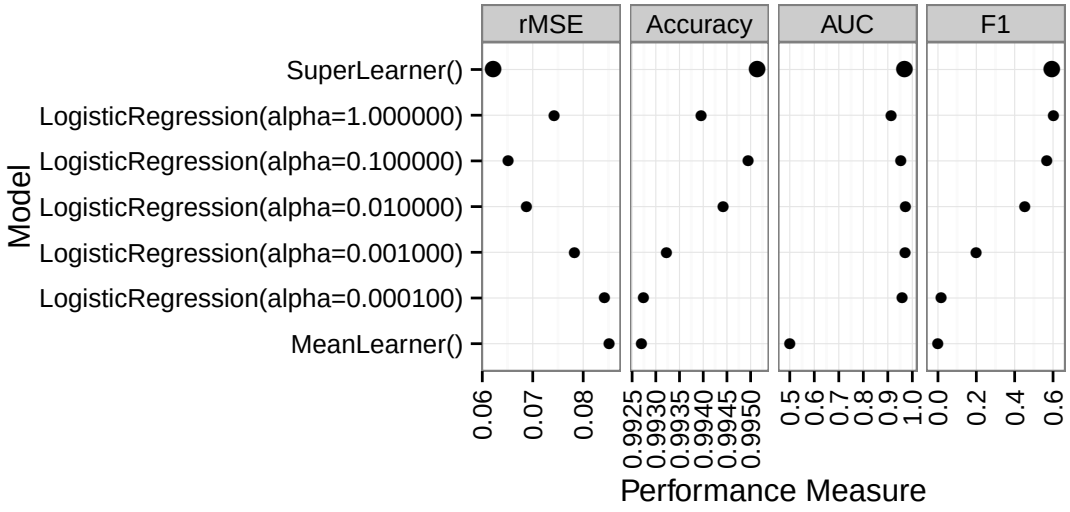

Figure 3: Validation performance for a collection of LogisticRegression learners with different learning rates (Wikipedia)

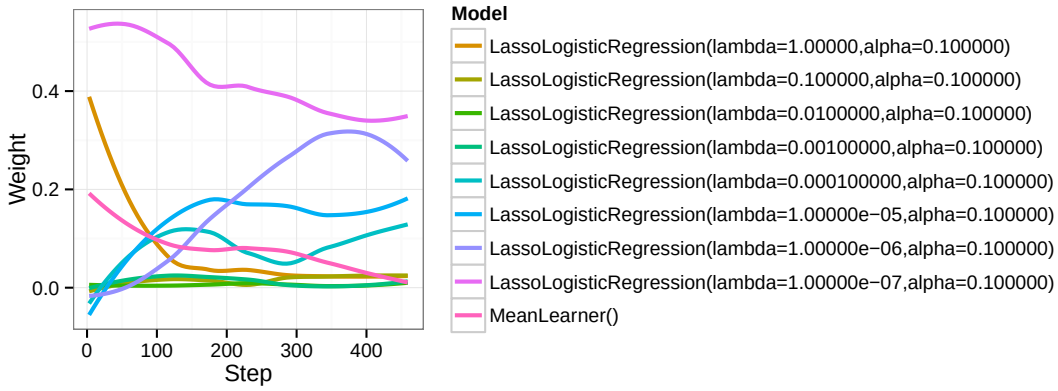

Figure 4: Weights for a collection of LassoLogisticRegression learners with different regularization parameters (Wikipedia)

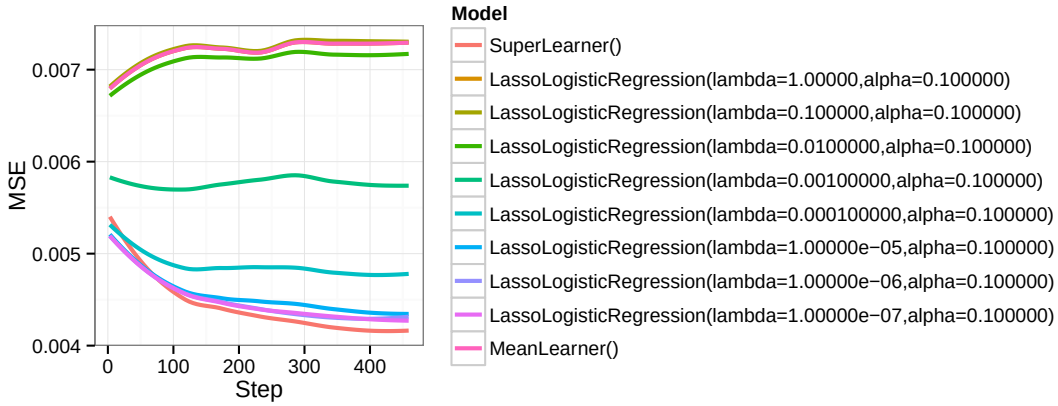

Figure 5: MSE for a collection of LassoLogisticRegression learners with different regularization parameters (Wikipedia)

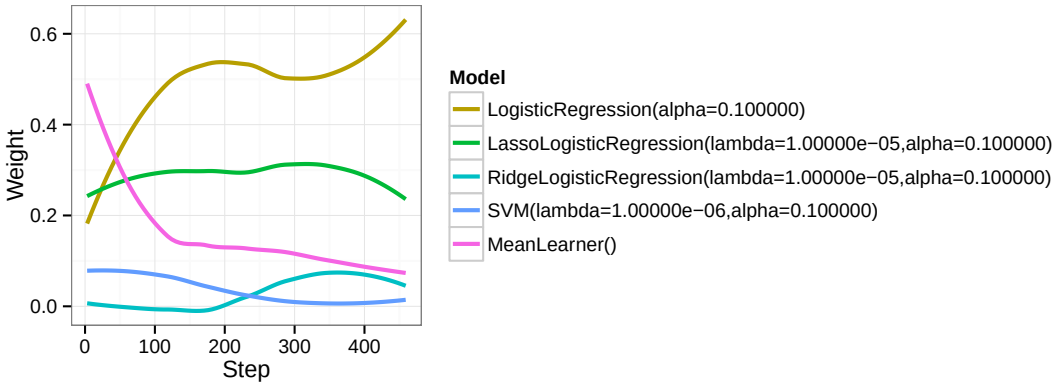

Figure 6: Weights for a collection of different learners (Wikipedia)

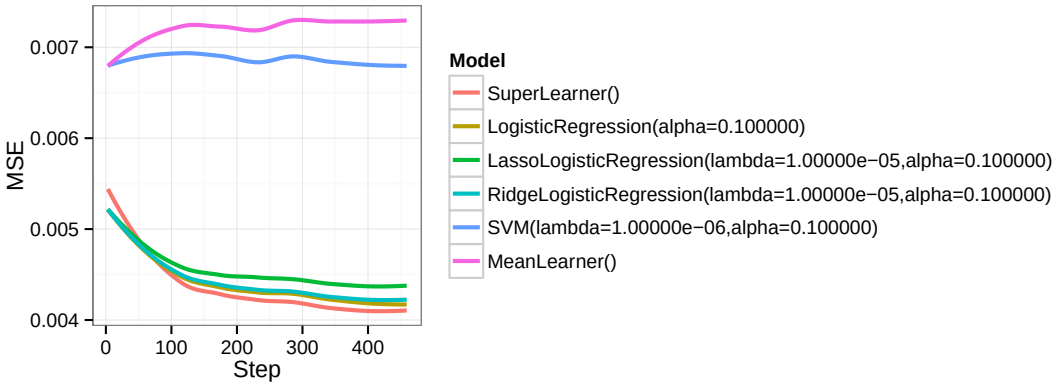

Figure 7: MSE for a collection of different learners (Wikipedia)

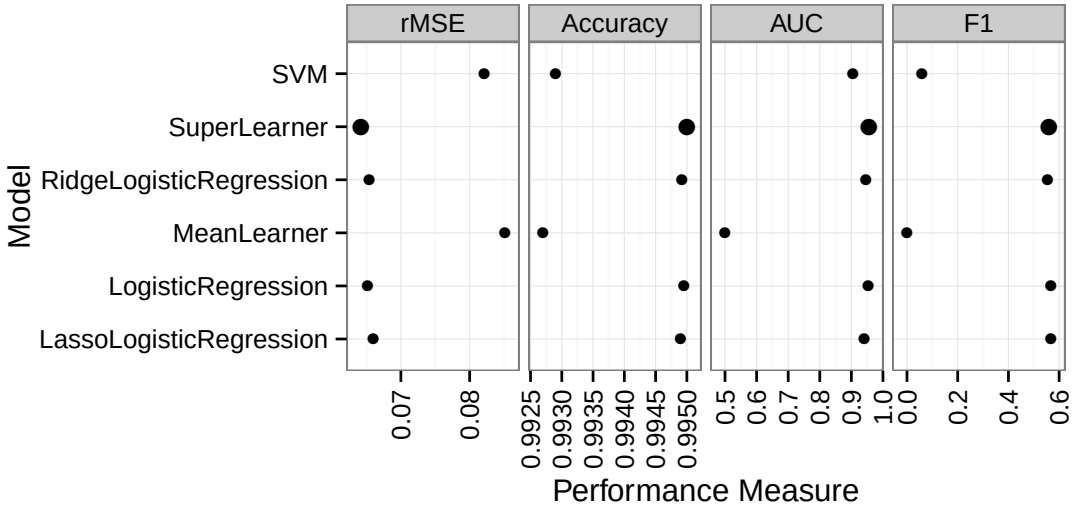

Figure 8: Validation performance for a collection of different learners (Wikipedia)

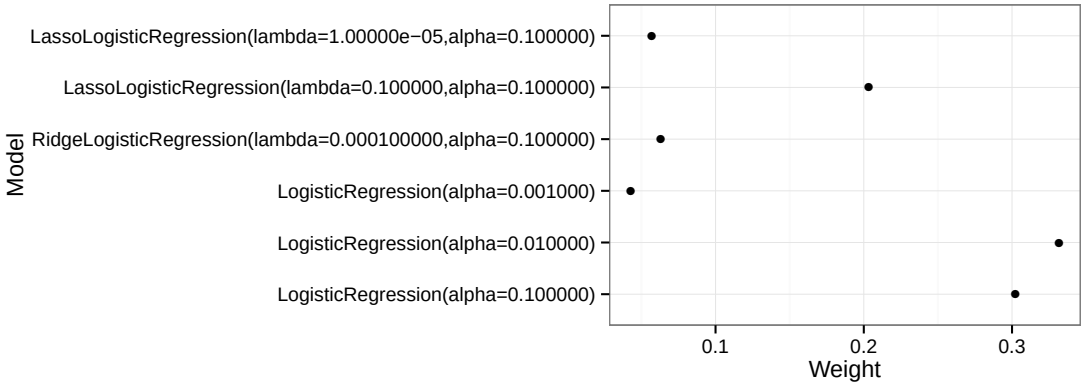

Figure 9: Final Weights for a collection of 96 different learners (Stack Overflow)

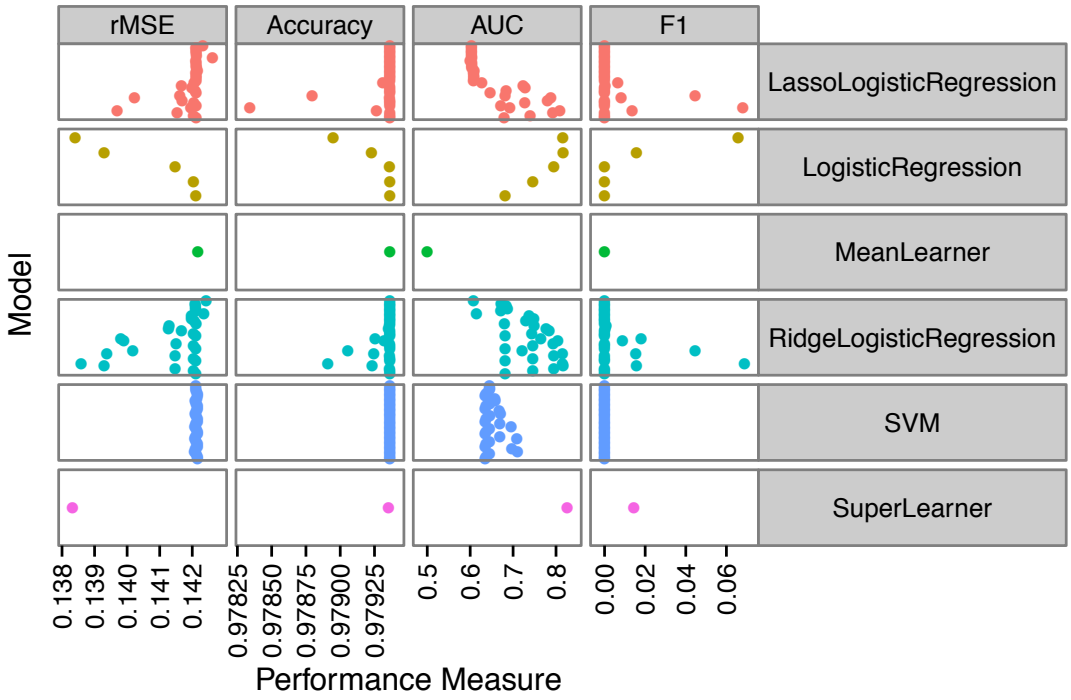

Figure 10: Validation performance for a collection of 96 different learners (Stack Overflow)

5 Lessons learned and future work

In completing this project, we saw that our implementation of SuperLearner performed as well as any individual algorithm in the library, as expected, while not overfitting even when the algorithm library was very large. Inclusion of algorithms in the SuperLearner library that were implemented with stochastic gradient descent also had the unforeseen advantage of providing an efficient way to "auto-tune" the learning rates of these algorithms by running many versions concurrently.

References

Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer, 2010.

John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12:2121–2159, 2010.

John Langford. LBFGS. Lecture notes from New York University course "Large Scale Machine Learning", February 2013. URL http://cilvr.cs.nyu.edu/diglib/lsml/lecture02-lbfgs.pdf.

Alexandru Niculescu-Mizil and Rich Caruana. Predicting good probabilities with supervised learning. In *Proceedings of the 22nd international conference on Machine learning*, pages 625–632. ACM, 2005.

Nicol Schraudolph, Jin Yu, and Simon Günter. A stochastic quasi-Newton method for online convex optimization. In *Proceedings of the* 11^{th} *International Conference on Artificial Intelligence and Statistics*

(AISTATS 2007), pages 433–440, San Juan, Puerto Rico, March 2007. Society for Artificial Intelligence and Statistics.

Mark J van der Laan, Eric C Polley, and Alan E Hubbard. Super learner. *Statistical Applications in Genetics and Molecular Biology*, 6(1):1–21, 2007.