# Supplementary Material for *Sstack*: An R Package for Stacking with Applications to Scenarios Involving Sequential Addition of Samples and Features.

## 1 Detailed Stacking Results

Gene Expression (GE) features and Area under the Dose-Response Curve (AUC) data have been retrieved from the Cancer Cell Line Encyclopedia (CCLE) [Barretina *et al.*, 2012] database for 490 samples. These samples are distributed into datasets using the diagram shown in figure 1. To simulate a condition in which sequential stacking is beneficial, features are distributed such that each Horizontal ($H_i$) model has an equal number while the samples are distributed using two separate schema. In the first schema, samples are evenly split among all Vertical ($V_i$) models while in the second schema, the lowest Vertical model ($V_n$) is biased such that it is has a larger portion of the samples. To estimate the error of our model we leave out 20% of the sample data for testing and use the remaining for training. After building both the individual and stacked models, the mean-square error (MSE) is calculated for the hold out test data. This process is repeated 100 times, randomly reassigning the features and samples to different groups. The final reported error is the average over all 100 iterations.

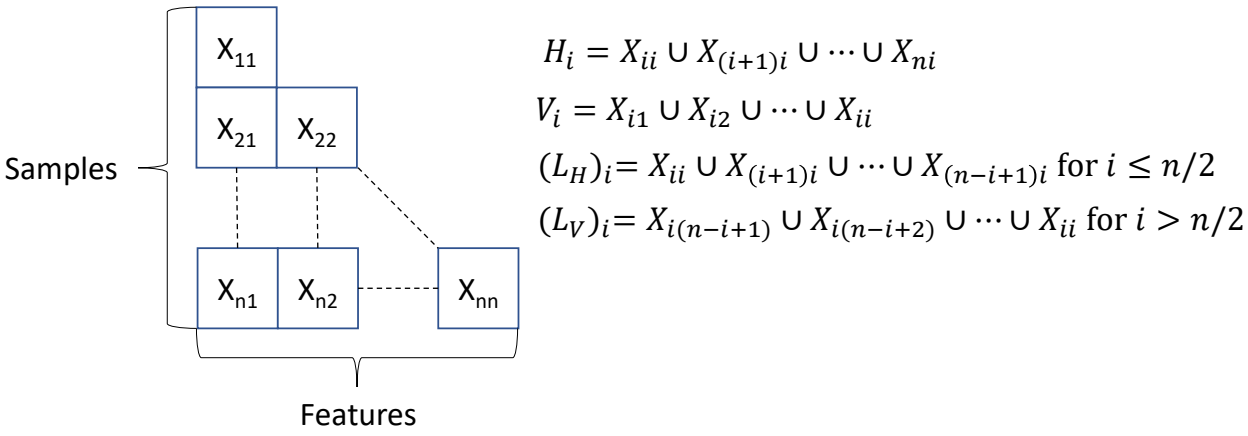



Figure 1: Illustration of Stacking Heterogeneous Data Sets Problem.

### 1.1 Bootstrap Stacking

We randomly but evenly divide the chosen features into 4 separate data sets shown in Figure 1. The number of bootstrap iteration, $N_{bs}$, is determined heuristically by randomly select 300 samples for training and then another 100 for testing. After estimating the error this process is repeated 500 times with a new training/testing set in each iteration before increasing $N_{bs}$ and then repeating the experiment. The mean-square error (MSE) estimates of the stacked model predictions for changing $N_{bs}$ is shown in figure 2. We note that horizontal stacking outperforms the other methods even for small $N_{bs}$ and MSE reaches a saturation point at around $N_{bs} = 25$. As such that an $N_{bs}$ of 25 is chosen for the remaining experiments utilizing CCLE alone.

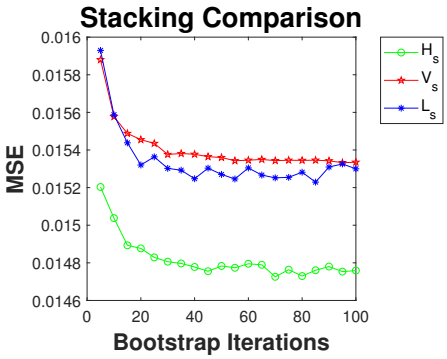
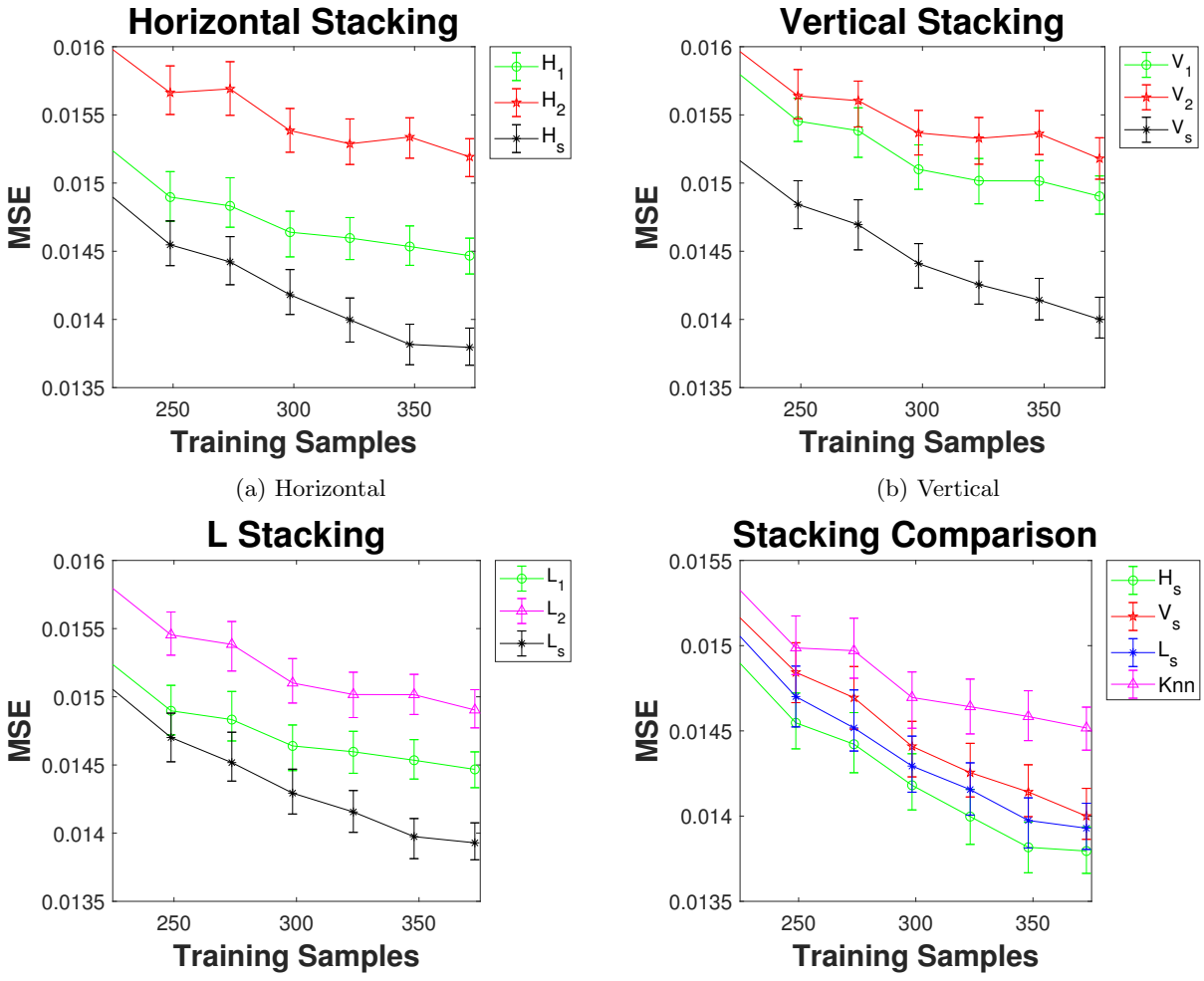



Figure 2: Mean square error (MSE) of 4-Layer Horizontal ($H_s$), Vertical ($V_s$), and L ($L_s$) Stacked Models with varying number of Bootstrap Iterations.

## 1.2 Stacking of Unbiased Datasets

We start our analysis by evenly splitting samples among the vertical datasets. For example, for $n = 2$, 400 training samples and 300 features, there would be two Horizontal datasets, one with a maximum size of $400 \times 150$ and the second with a size of $200 \times 150$ while the two Vertical datasets have a size of $200 \times 150$ and $400 \times 300$.

Results for $n = 2$ layer stacking is shown in figure 3. The sub figures show the estimated MSE for each type of individual ($H1$, $V1$, etc.) model as well as the different methods of stacking associated with the corresponding set. In addition, sub-figure (d) provides a comparison of all the different methods of stacking as well as a comparison with imputing the missing data with the KNN impute method [Liew *et al.*, 2011]. For 2-layer stacking, the stacked model always provides an improvement to model accuracy, even with a relatively small number of training samples. Additional comparisons for 4 and 6 layer stacking is given in figures 4 and 5 respectively. Unlike in 2 layer stacking, we note that stacking does not always provide an improvement in model accuracy without a sufficient number of training samples. The amount will heavily depend on the number of datasets being stacked, for example in 4 layer stacking (figure 4a) improvement can not be inferred until at least 280 training samples while in 6 layer stacking (figure 5a) approximately 380 samples are needed.



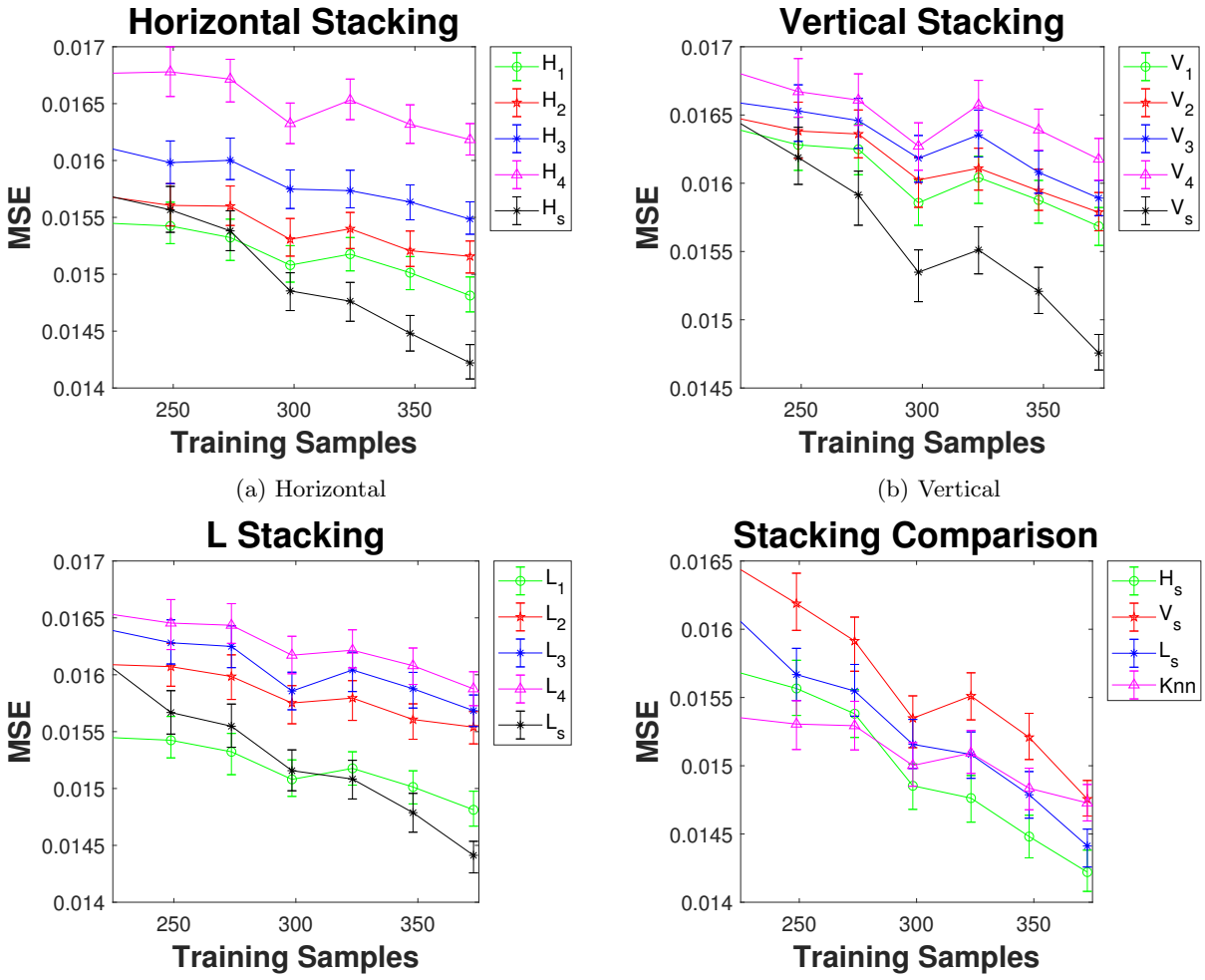

(a) Horizontal

(b) Vertical

(c) L

(d) All Stacked

Figure 3: Mean-square Error analysis of individual stacking methods (a-c) as well as a comparison of stacked models verses KNN impute (d) for 2-layer stacking of CCLE data. Samples are distributed evenly across the layers.



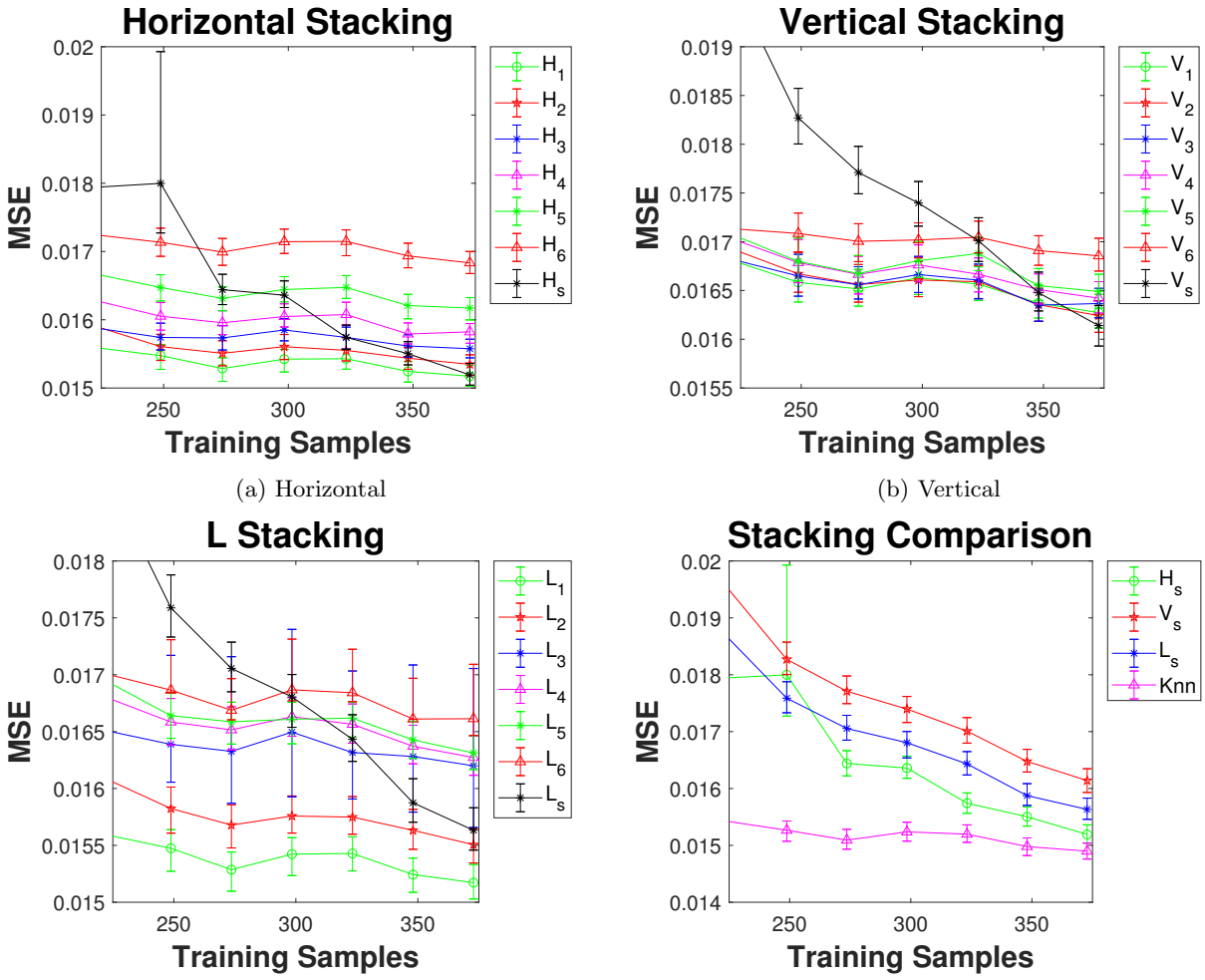

(a) Horizontal

(b) Vertical

(c) L

(d) All Stacked

Figure 4: Mean-square Error analysis of individual stacking methods (a-c) as well as a comparison of stacked models verses KNN impute (d) for 4-layer stacking of CCLE data. Samples are distributed evenly across the layers.




(a) Horizontal

(b) Vertical

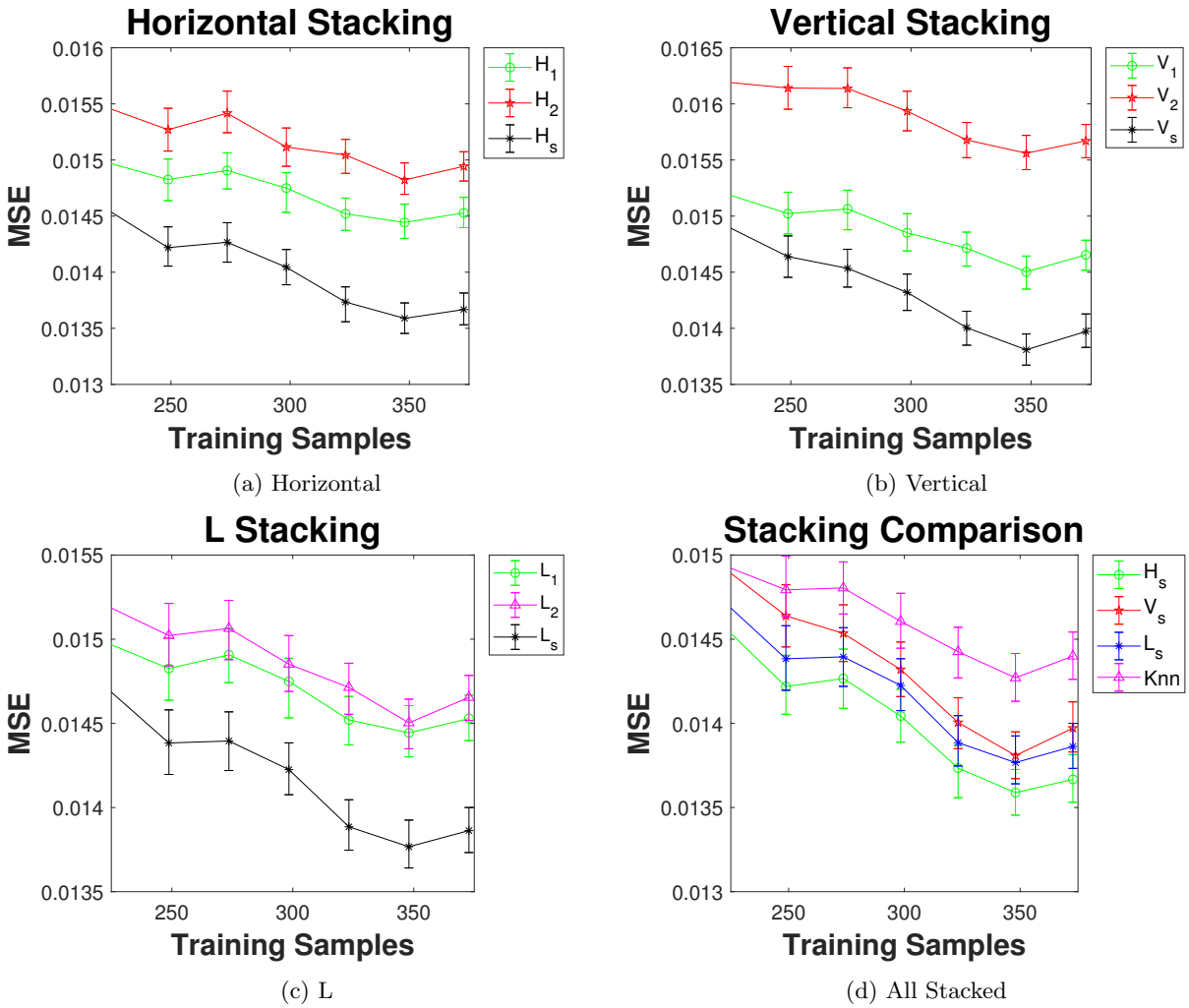
(c) L

(d) All Stacked

Figure 5: Mean-square Error analysis of individual stacking methods (a-c) as well as a comparison of stacked models verses KNN impute (d) for 6-layer stacking of CCLE data. Samples are distributed evenly across the layers.

### 1.3 Stacking of Biased Datasets

In contrast to section 1.2, the datasets have been biased in this section such that every vertical layer, with the exception of the lowest layer, contains $1/(n+1)$ fraction of the samples while the lowest layer contains the remaining $2/(n+1)$ fraction of the samples. As a consequence of this the weights are generated with a larger amount of samples, improving the number accuracy of the stacked model. To compare with the previous example, for $n = 2$ and 400 training samples, our Vertical datasets would be of size of $133 \times 150$ and the second with a size of $266 \times 300$. With this partitioning method the number of samples used for building our linear stacked model increases by a factor of $2n/(n+1)$. The results for this type of partitioning for 2, 4, and 6 layer stacking is given in figures 6, 7, and 8 respectively. As anticipated, stacking of these biased datasets provide a greater increased to performance when compared to the unbiased datasets. We see that stacking provides improvement at a lower number of training samples and that the overall mean-square error of the stacked models is lower. For example if we consider Horizontal stacking, in figure 5a we note that there is no benefit to stacking even at the maximum number of training samples but in figure 8a we see an approximate 10% decrease in the MSE. Finally, we take note that Horizontal stacking still consistently outperforms the other methods of stacking.



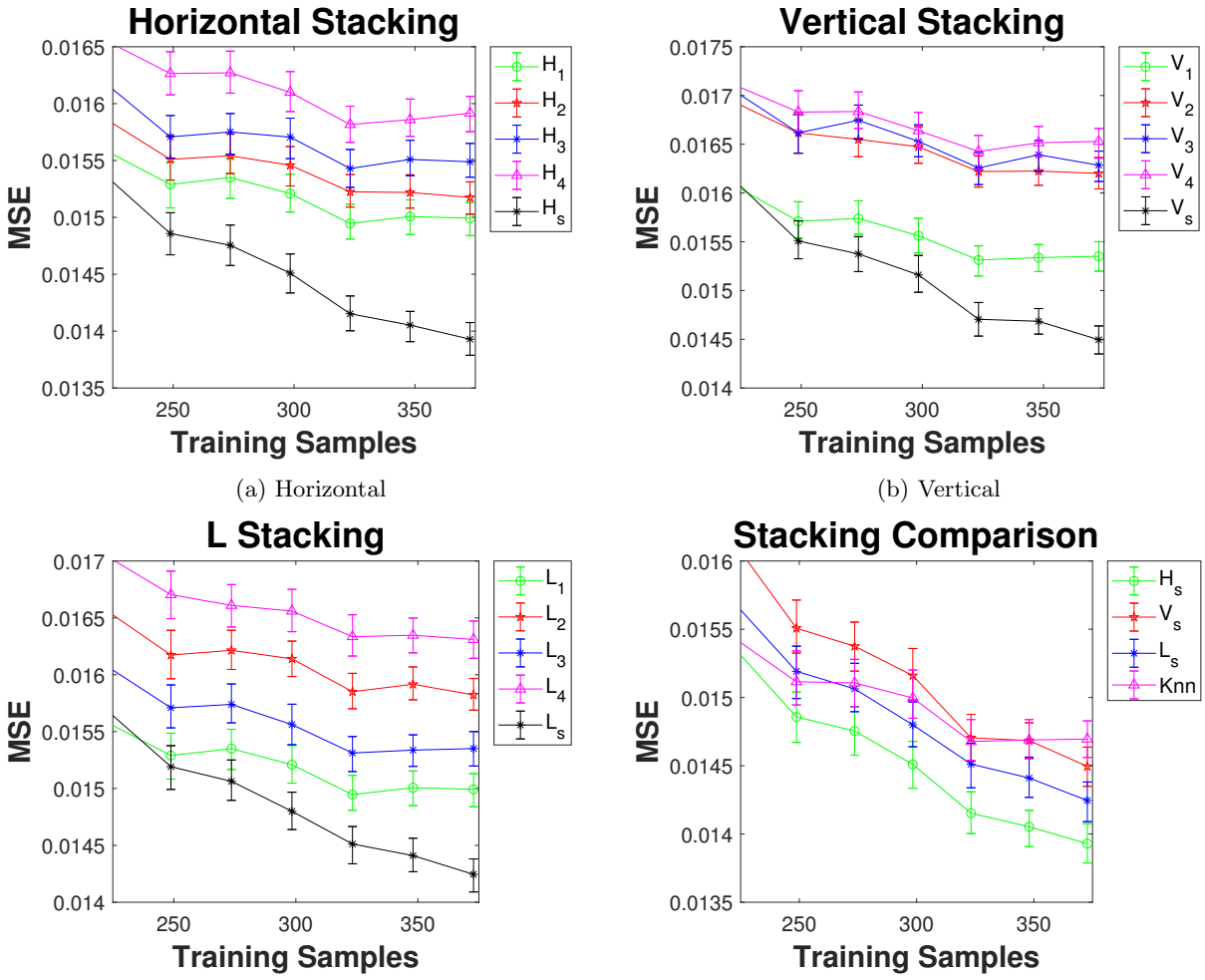

(a) Horizontal

(b) Vertical

(c) L

(d) All Stacked

Figure 6: Mean-square Error analysis of individual stacking methods (a-c) as well as a comparison of stacked models verses KNN impute (d) for 2-layer stacking of CCLE data. Samples are biased such that the top vertical layer has an increased number of samples compared to the remaining layers.



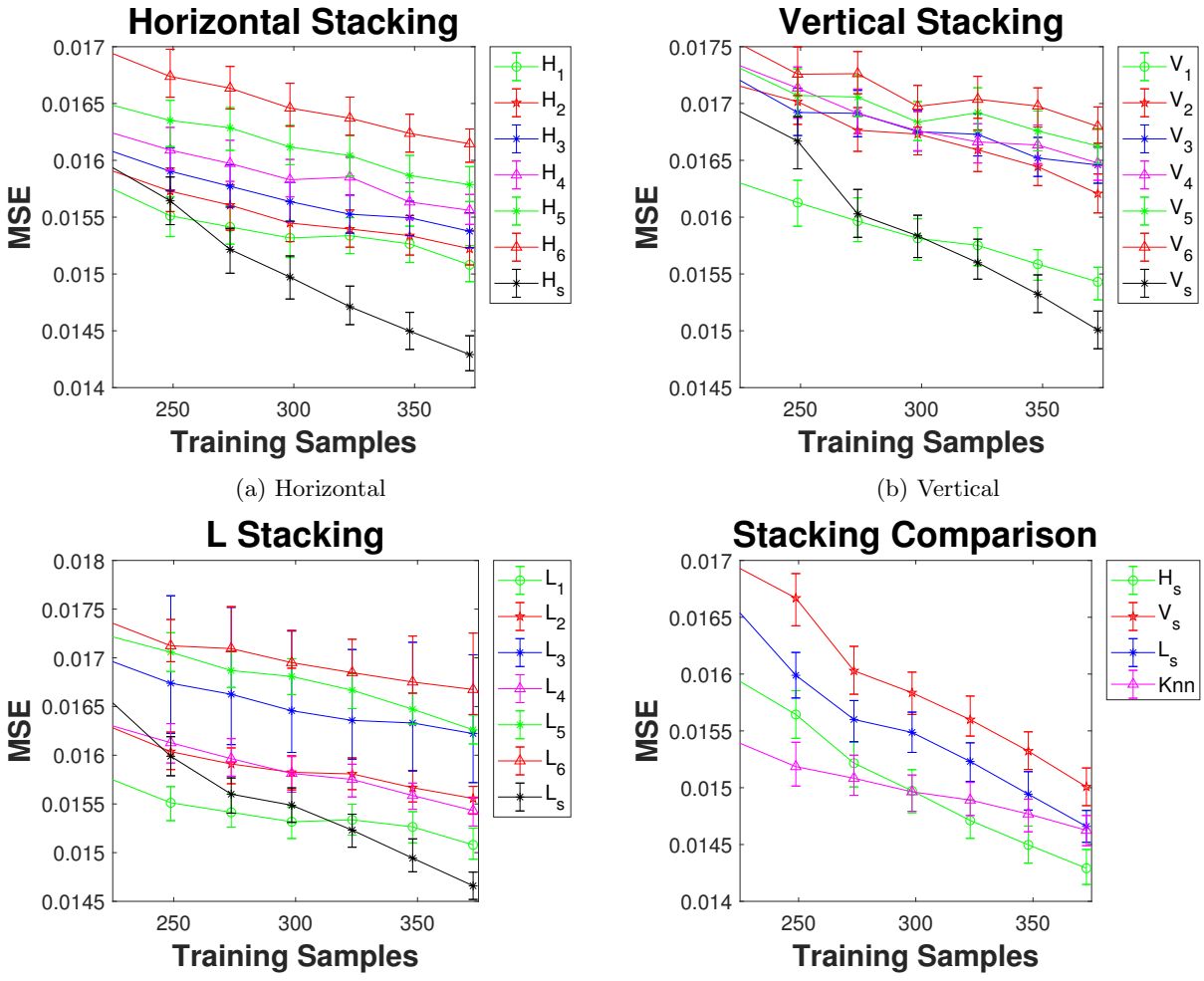

(a) Horizontal

(b) Vertical

(c) L

(d) All Stacked

Figure 7: Mean-square Error analysis of individual stacking methods (a-c) as well as a comparison of stacked models verses KNN impute (d) for 4-layer stacking of CCLE data. Samples are biased such that the top vertical layer has an increased number of samples compared to the remaining layers.




(a) Horizontal

(b) Vertical

(c) L

(d) All Stacked

Figure 8: Mean-square Error analysis of individual stacking methods (a-c) as well as a comparison of stacked models verses KNN impute (d) for 6-layer stacking of CCLE data. Samples are biased such that the top vertical layer has an increased number of samples compared to the remaining layers.

## 2 Example of Stacking GE and RPPA Data

To show a more practical example of stacking, we next stack GE and RPPA data. From both the GE and RRPA databases, we run RELIEFF to pick the top 150 features. We built the individual and stacked models using $N_{bs} = 50$ to generate the weights. For error estimation, we perform Leave-One-Out cross validation on the 194 cell lines that have GE and RPPA measurements. Results of performing this stacking is shown in table 1. We see that all forms of stacking outperforms the best individual model. In addition, we observe as before that Horizontal stacking outperforms the other two methods of stacking. Finally, simply estimating the RPPA features with KNN Impute provides no benefit to prediction accuracy.

Table 1: Mean absolute error of predictions from combining GE and RPPA data to predict AUC response for the drug 17-AAG. Results for individual and stacked models as well as a KNN Impute baseline is shown. Error estimated utilizing Leave-One-Out validation on a set of 194 overlapping samples.

| | Horizontal | Vertical | L | KNN Impute |
|---|---|---|---|---|
| H1/V1/L1 | 0.7826 | 0.8159 | 0.7741 | NA |
| H2/V2/L2 | 0.8399 | 0.7797 | 0.8122 | NA |
| Stacked | 0.7381 | 0.7436 | 0.7440 | 0.7885 |

## 3 Package Application Example

In this section we demonstrate how a user may utilize the *Sstack* package with a example. This code stacks two sets of GE data to predict the AUC for all samples that have been tested with the drug *17-AAG*. Stacking is simulated using an unbiased 2-layer model. The error for both individual and stacked models are estimated using a held out set of sample ($Xt$). This data and sample code is included in the package.

```
# Load in Package and Data
library(Sstack)
library(doParallel)
data(StackData)

# Form difference datasets, on with all samples while the other only has half the samples.
AUC=StackData[[1]]
GE=StackData[[2]]
X1 <- GE[1:400,:]
X2 <- GE[200:400,76:150]
Xt <- GE[401:487,]

# Random seed for repeatability
set.seed(1)

# Register parallel cluster, optional but greatly increases speed.
cl <- makeCluster(2)
registerDoParallel(cl)

# Build the model with 100 trees and 50 Bootstrap iterations.
Hbs <- BSstack(T = 100, iter = 50, Y = AUC, X1 = X1, X2 = X2)

# Predict on the test set and measure mean absolute error.
Yp <- BSstack_predict(Hbs[[1]],Xt)
maeH1 <- mean(abs(AUC[401:487,]-Yp[,1]))
maeH2 <- mean(abs(AUC[401:487,]-Yp[,2]))
maeHs <- mean(abs(AUC[401:487,]-Yp[,3]))
```

# 4 Parameter Selection Guidelines

Table 2 provides a summary of the major parameters that should be inputted in our stacking algorithm. The first set ($T$, *mtry*, and *nodesize*) are the typical parameters found in a Random Forest implementation. For these parameters, the user may submit only a single value in which case that parameter will apply to all of the Random Forests that are being stacked; or the user can submit a list of values in which case each Random Forest will be built with separate parameters. An additional parameter, *iter*, controls the number of bootstrap-sampled stacked models used to estimate the weights. Finally, an optional cross-validation ($CV$) parameter may be given in order to automatically estimate the error of both the individual and stacked models utilizing the samples with full record.

Table 2: List of parameters utilized in the *Sstack* package. The variable type, range of values, default values and a brief description is given.

| Parameter | Type | Typical Range | Default | Description |
|---|---|---|---|---|
| T | int | [50 500] | 100 | Number of trees used in building each Random Forest. |
| mtry | int | [5 250] | $nFeats/3$ | Number of variables used for splitting at each tree node. |
| nodesize | int | [1 10] | 5 | Minimum size of terminal nodes. |
| iter | int | [10 200] | 100 | Number of bootstrap samples to take in order to estimate the stacking weights. Large number produced more accurate model but with high training time. |
| CV | int | [0 $nSamp$] | NAN | Number of folds to separate the overlapping samples in order to perform cross-validation. If 0 is given, then leave-one-out validation is performed instead. If no value is given, then no cross-validation is performed. |

# 5 Runtime Comparisons

To measure the computational complexity of our package, we compare how the mean runtime of our stacked models compare to just utilizing KNN impute to build a single Random Forest (RF) model. We perform our experiments by stacking the $490 \times 150$ GE data with the $194 \times 150$ RPPA data. Tests were performed on a desktop computer with an Intel Core i7-7700 3.60GHz CPU and 16GB of RAM and the results are shown in table 3. As evident from this table, we see that the average runtime of our stacked model is approximately $n * N_{bs}/nCore$ where $nCore$ is the number of cores available. The package has been parallelized using R's 'parallel' package but the user must first register the workers beforehand or only a single core implementation will be used. Future implementations of the *Sstack* package could include the use of cluster computing frameworks such as Apache Spark [Zaharia *et al.*, 2016] to further reduce the runtime.

Table 3: Mean runtime of stacked models. Estimated taking the average runtime of 194 models created by stacking a $490 \times 150$ GE dataset with a $194 \times 150$ RPPA dataset.

| Cores | nSamp (X1/X2) | $N_{bs}$ | nTrees | Horizontal | Vertical | L | KNN Impute |
|---|---|---|---|---|---|---|---|
| 2 | (490/194) | 100 | 100 | 59.0s | 47.8s | 69.7s | 1.26s |
| 4 | (490/194) | 100 | 100 | 35.7s | 29.0s | 42.6s | 1.25s |
| 8 | (490/194) | 100 | 100 | 27.8s | 22.5s | 32.7s | 1.47s |
| 8 | (490/194) | 50 | 100 | 14.8s | 12.0s | 17.3s | 1.51s |
| 8 | (490/194) | 50 | 50 | 7.53s | 6.24s | 9.10s | 0.67s |
| 8 | (290/94) | 100 | 100 | 17.6s | 15.3s | 20.0s | 1.03s |

# 6 Analysis of Stacking

In this section we offer a heuristic reasoning on why we expect the Horizontal stacking to perform better than its Vertical counterpart.

Denote the RF training dataset as $\mathcal{D}_F = (Y, \boldsymbol{x})$. We can view RF prediction as a weighted average of the individual tree prediction, i.e.,

$$\bar{Y}(\boldsymbol{x}) = \frac{1}{T}\sum_{j=1}^{T}\sum_{i=1}^{n} w_i(\boldsymbol{x}, \Theta_j) y(i) \tag{1}$$

Define the random variable $Z_j(\boldsymbol{x}, \Theta_j) = \sum_{i=1}^{n} w_i(\boldsymbol{x}, \Theta_j) y(i), \quad j = 1, 2..., T$ as the prediction obtained from the $j$th tree generated by the $\Theta-$ process. Let us assume that the $\Theta-$ process induces a valid distribution on the finite collection

$$\boldsymbol{Z}(\boldsymbol{x}, \Theta) = [Z_1(\boldsymbol{x}, \Theta_1), Z_2(\boldsymbol{x}, \Theta_2), ..., Z_T(\boldsymbol{x}, \Theta_T)] \tag{2}$$

Now, observe that each tree attempts to predict the target $\mu(\boldsymbol{x}) = E(Y|\boldsymbol{x})$ and the RF predictor, $\bar{Y}$ (obtained in 1), emerges as the sample average, $\bar{Z}$ of $\boldsymbol{Z}(\boldsymbol{x})$. However, finite sample tree predictions are biased [Zhang and Lu, 2012] resulting in $E(Z_j(\boldsymbol{x})) = \alpha_j(n) + \beta_j(n)\mu(\boldsymbol{x})$ and $Var(Z_j(\boldsymbol{x})) = \sigma_j^2 \quad j = 1, 2, .., T$, where the additive and multiplicative biases ($\alpha_j(n)$, $\beta_j(n)$, respectively) disappear as $n \to \infty$ under some smoothness condition on true $\mu(\boldsymbol{x})$ [Biau, 2012]. Note that, in this construction $\sigma_j^2$ can be interpreted as the variance of individual tree estimates and, therefore, is of the order $k_n/n$ where $k_n$ is approximately the number of terminal nodes and $n$ is the number of samples on which the tree is built [Devroye *et al.*, 1996].

For illustration purpose, we assume $\alpha_j = 0$, $\beta_j = \beta > 0$ and $\sigma_j^2 = \sigma^2, \quad j = 1, 2, ..., T$. For notational simplicity, we suppress the arguments $n$,$\Theta$ and $\boldsymbol{x}$ in relevant statistics henceforth. Under the assumption of Gaussianity and conditional independence, the joint distribution of $[\boldsymbol{Z}|\mu, \beta, \sigma^2]$ is $\prod_{j=1}^{T} Normal(\beta\mu, \sigma^2)$. Next, suppose we have another model, $M$, potentially operating on a different set of inputs, $\boldsymbol{x}_m$, but predicting the same response variable $Y$. We denote the training data for this model $M$ as $\mathcal{D}_M$. The output of this model is $\mu_m$ which is an estimator of $E(Y|\boldsymbol{x}_m)$. Now to pool both RF and model $M$ together to generate predictions of $Y$, we impose a $Normal(\mu_m, \tau^2)$ prior on $\mu$. If $M$ is another ensemble model, $\tau^2$ can be computed in the same vein as $\sigma^2$.

Therefore the hierarchical specification takes the form

$$[\boldsymbol{Z}|\mu, \sigma^2, \beta][\mu|\mu_m, \tau^2][\sigma^2, \tau^2, \beta] \tag{3}$$

and the conditional posterior distribution of $\mu$, $[\mu|\sigma^2, \tau^2, \beta, \mathcal{D}_F, \mathcal{D}_M]$, turns out to be Normal$\big(\lambda, \nu^2\big)$, where,

$$\nu^2 = \left(\frac{1}{\tau^2} + \frac{T\beta^2}{\sigma^2}\right)^{-1}, \tag{4}$$

$$\lambda = \frac{T\beta\tau^2}{\sigma^2 + T\beta^2\tau^2}\bar{Z} + \frac{\sigma^2}{\sigma^2 + T\beta^2\tau^2}\mu_m. \tag{5}$$

Note that, the Bayes estimate under square error loss is the posterior mean $\lambda$ which happens to have similar form as the linear stacking estimator.

Observe that if $\sigma^2 \ll \tau^2$ and $\beta > 1$ then $\lambda \approx \frac{1}{\beta}\bar{Z}$. Thus, when the ensembles in RF overpredicts, the stacking estimator downweighs the RF estimator (with negligible contribution from $\mu_m$) thereby reducing the bias. On the other hand, if $\sigma^2 \ll \tau^2$ and $0 < \beta < 1$ then $\lambda \approx \frac{C\beta}{1+C\beta^2}\bar{Z} + \frac{1}{1+C\beta^2}\mu_m$, where $C = T\tau^2/\sigma^2 \gg 1$. In this situation RF ensemble underpredicts but stacking operation counteracts in the following way: (a) When $C\beta \leq 1$, the stacking estimate underweighs RF estimate but adds a non-trivial fraction of $\mu_m$. In an extreme situation, when $\beta(\in \mathbb{R}^+)$ is in the neighborhood of 0, the stacking estimator does not put any weight on the RF estimate and solely uses $\mu_m$ as the prediction, thereby reducing the RF bias. (b) When $C\beta \gg 1$ the stacking estimator upweighs the RF estimate with minimal contribution from $\mu_m$. Clearly, in all the three foregoing situation, stacking helps reducing the bias of RF estimates.

What happens when $\sigma^2$ and $\tau^2$ are comparable or $\sigma^2 \gg \tau^2$? Our argument from the previous paragraph suggests that the debiasing characteristic of stacking operation will critically hinge on $T$. However, arbitrarily large $T$ is not useful because after a certain number of trees, individual tree outputs will be correlated hence

violating the fundamental premise of conditional independence in our setup. Consequently, the effect of stacking operation on debiasing RF output is ambiguous.

The fact that we need $\sigma^2 \ll \tau^2$ to force the stacking estimator operate as a debiasing devise indicates that we ought to design the stacking operation in such a way that the above condition is satisfied. Consider a generic situation where $\mathcal{D}_M$ consists of $n_1$ independent samples and the feature matrix $\boldsymbol{x}_m$ is of dimension $n_1 \times p_1$. $\mathcal{D}_F$ consists on $n_2$ samples and the corresponding feature matrix $\boldsymbol{x}$ is of dimension $n_2 \times (p_1 + p_2)$, such that $\mathcal{D}_F$ includes all the features observed in $\mathcal{D}_M$ and also $p_2$ additional features. Since the additional set of features available in $\mathcal{D}_F$ may also contain important predictors, we must predict the response utilizing the entire set of $p_1 + p_2$ features. One can easily combine these two training sets by simply stacking RFs trained on $\mathcal{D}_M$ and $\mathcal{D}_F$. This operation is nothing but *vertical stacking*. Since both are RF estimators $\sigma^2$ is of order $k_./n_2$ and $\tau^2$ is of order $k_./n_1$. Since $k_.$ is typically user specifed and can be made to remain constant in both RFs, the variances of the stacking components are essentially determined by the sample sizes of the respective training set. Clearly, if $n_2 < n_1$ the above condition relating the variances of the stacking components cannot be enforced. One can switch the generic label $\sigma^2$ and $\tau^2$, but in this situation the stacking operation would be more effective in debiasing the RF estimates obtained from $\mathcal{D}_M$- which is counter-productive because $\mathcal{D}_F$ contains more information as compared to $\mathcal{D}_M$ and hence we would like to put more weight on the RF trained on $\mathcal{D}_F$ than the other way around.

With our conceptualization of *horizontal stacking*, we can enforce the variance condition regardless of the sample sizes of $\mathcal{D}_M$ and $\mathcal{D}_F$. We first partition the feature matrix associated with $\mathcal{D}_F$ into two parts $\boldsymbol{x}^{n_2 \times (p_1+p_2)} = (\boldsymbol{x}_{p_1}^{n_2 \times p_1}, \boldsymbol{x}_{p_2}^{n_2 \times p_2})$. We then train an RF on $n_1 + n_2$ samples with feature matrix $(\boldsymbol{x}_m, \boldsymbol{x}_{p_1})'$. If $\sigma^2$ is the variance associated with this stacking component, then $\sigma^2$ is of the order $k_./(n_1 + n_2)$. The other model is also an RF but trained on $n_2$ samples and feature matrix $\boldsymbol{x}_{p_2}$. If $\tau^2$ is the variance associated with this stacking component, then $\tau^2$ is of the order $k_./n_2$. Keeping the number of terminal node constant, we can easily see $\sigma^2 < \tau^2$ and hence we expect *horizontal stacking* to be more efficient in debiasing than *vertical stacking*. Furthermore, as $n_1, n_2 \to \infty$ and $\beta \to 1$, it is easy to see, from (5), that the variance of horizontal stacking estimator is smaller than its vertical counterpart.

# References


[Barretina *et al.*, 2012] Barretina,J. *et al.* (2012) The Cancer Cell Line Encyclopedia enables predictive modelling of anticancer drug sensitivity. *Nature,* **483** (7391), 603–607.

[Biau, 2012] Biau,G. (2012) Analysis of a random forests model. *J. Mach. Learn. Res.,* **13** (1), 1063–1095.

[Devroye *et al.*, 1996] Devroye,L., Gyorfi,L. and Lugosi,G. (1996) *A Probabilistic Theory of Pattern Recognition.* Springer, Berlin.

[Liew *et al.*, 2011] Liew,A.W.C., Law,N.F. and Yan,H. (2011) Missing value imputation for gene expression data: computational techniques to recover missing data from available information. *Briefings in Bioinformatics,* **12** (5), 498–513.

[Zaharia *et al.*, 2016] Zaharia,M., Xin,R.S., Wendell,P., Das,T., Armbrust,M., Dave,A., Meng,X., Rosen,J., Venkataraman,S., Franklin,M.J., Ghodsi,A., Gonzalez,J., Shenker,S. and Stoica,I. (2016) Apache spark: a unified engine for big data processing. *Commun. ACM,* **59** (11), 56–65.

[Zhang and Lu, 2012] Zhang,G. and Lu,Y. (2012) Bias-corrected random forests in regression. *Journal of Applied Statistics,* **39** (1), 151–160.