Pages

Alessia

2023-06-15

INTRODUCTION

In a typical prediction problem, there is a trade-off between two factors: bias and variance. Bias refers to the error introduced by simplifying assumptions in a model, while variance refers to the amount by which predictions vary for different samples of data.

When we try to improve the fit of a model, there comes a point where increasing the precision of the fit leads to higher variance in predictions for future observations. On the other hand, reducing prediction variance can result in a higher expected bias for future predictions.

To address this trade-off, Breiman (1996) introduced a technique called bagging, which stands for "Bootstrap AGGregatING." The idea behind bagging is simple. Instead of relying on a single model fit to the observed data for making predictions, multiple bootstrap samples (randomly selected subsets) of the data are created. Each of these samples is used to fit a model, and the predictions from all the fitted models are averaged to obtain the bagged prediction.

Breiman explains that bagging works particularly well for modeling procedures that are sensitive to small changes in the data, such as classification and regression trees (CART). He also provides a theoretical explanation of how bagging reduces the mean-squared prediction error for such unstable procedures.

Unstable classifiers, such as trees, characteristically have high variance and low bias. Stable classifiers have low variance, but can have high bias.

Instability is an essential ingredient for bagging to improve accuracy.

We have a learning set (L) with data points $\{(y_n, x_n)\}$ for training a predictor $(\delta(x, L))$.

To improve the predictor, we use a sequence of learning sets $\{L_k\}$ drawn from the same distribution as L. We only have access to the predictors $\{\delta(x, L_k)\}$.

For numerical responses, we average the predictors $\delta(x, L_k)$ to get an aggregated predictor $\delta_A(x)$.

For class labels, we use voting among the predictors $\delta(x, L_k)$ to form the aggregated predictor $\delta_A(x)$.

If we don't have multiple learning sets, we mimic the process using repeated bootstrap sampling from L to create bootstrap replicates $\{L^{(B)}\}$. We form predictors $\delta(x, L^{(B)})$ using these replicates.

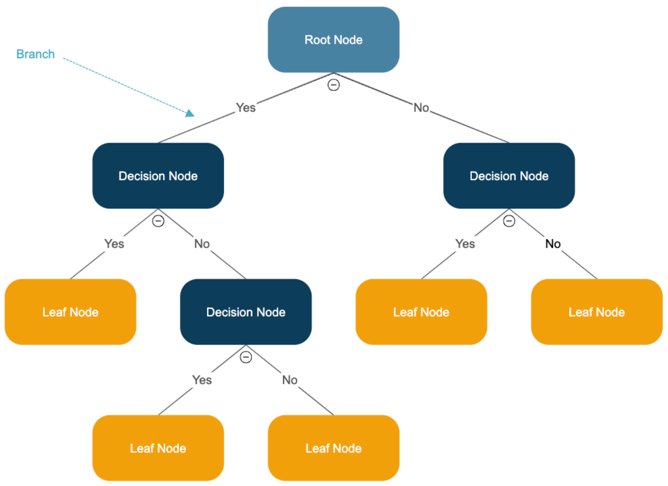
For numerical responses, the aggregated predictor is $\delta_B(x)$, which is the average of the predictors $\delta(x, L^{(B)})$.

For class labels, the aggregated predictor $\delta_B(x)$ is determined by voting among the predictors $\delta(x, L^{(B)})$.

This procedure is called bootstrap aggregating or bagging, where the bootstrap replicates $\{L^{(B)}\}$ are drawn to approximate the underlying distribution of L.

CLASSIFICATION TREE

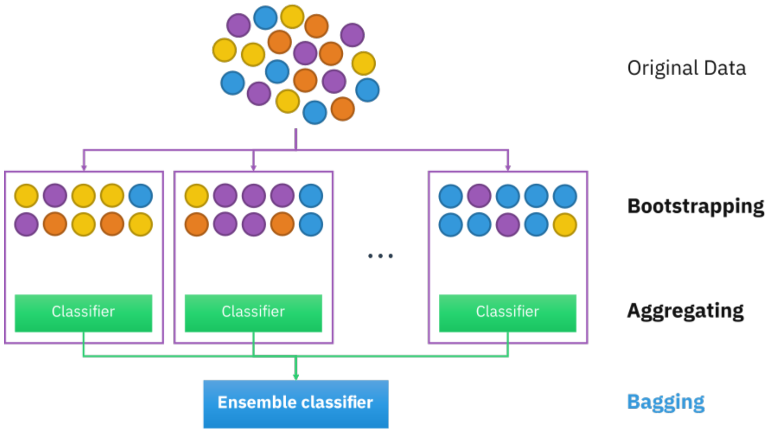
A classification tree is a structural mapping of binary decisions that lead to a decision about the class (interpretation) of an object (such as a pixel). Although sometimes referred to as a decision tree, it is more properly a type of decision tree that leads to categorical decisions. A regression tree, another form of decision tree, leads to quantitative decisions.



A classification tree is composed of branches that represent attributes, while the leaves represent decisions. In use, the decision process starts at the trunk and follows the branches until a leaf is reached.For a classification tree, we predict that each observation belongs to the most commonly occurring class of training observations in the region to which it belongs. In interpreting the results of a classification tree, we are often interested not only in the class prediction corresponding to a particular terminal node region, but also in the class proportions among the training observations that fall into that region. The task of growing a classification tree is quite similar to the task of growing a regression tree. Just as in the regression setting, we use recursive binary splitting to grow a classification tree. However, in the classification setting, RSS (Residual Sum Square) cannot be used as a criterion for making the binary splits. The classification tree suffer from high variance. This means that if we split the training data into two parts at random, and fit a classification tree to both halves, the results that we get could be quite different. In contrast, a procedure with low variance will yield similar results if applied repeatedly to distinct data sets; linear regression tends to have low variance, if the ratio of n to p is moderately large

Bagging

The bootstrap is used in many situations in which it is hard or even impossible to directly compute the standard deviation of a quantity of interest.Here the bootstrap can be used in order to improve statistical learning methods such as decision trees. The decision trees suffer from high variance. Bootstrap aggregation, or bagging, is a general-purpose procedure for reducing the variance of a statistical learning method; we introduce it here because it is particularly useful and frequently used in the context of decision trees.



Given a set of n n independent observations $Z_1, \ldots, Z_n$, each with variance σ^2, the variance of the mean $\overline{Z}$ of the observations is given by σ^2/n. In other words, averaging a set of observations reduces variance. Hence a natural way to reduce the variance and increase the test set accuracy of a statistical learning method is to take many training sets from the population, build a separate prediction model using each training set, and average the resulting predictions.

Random forest

Random forests are an improvement over bagged trees that aim to make the trees less similar to each other. In bagging, we create multiple decision trees using different training samples. However, in random forests, when building these trees, we make a small change. Instead of considering all predictors at each split, we randomly select a subset of predictors and choose only one from that subset for the split. We refresh this subset for each split, usually picking the square root of the total number of predictors.

This tweak is done because in bagged trees, if there is a strong predictor in the dataset, most or all of the trees will use that predictor in the top split. As a result, the predictions from these trees will be highly correlated. Unfortunately, averaging highly correlated predictions doesn't reduce the variability as much as averaging uncorrelated predictions. Therefore, bagging doesn't significantly reduce the variance compared to a single tree in such cases.

Random forests solve this problem by restricting each split to consider only a subset of predictors. This means that, on average, a portion of the splits won't even consider the strong predictor. By doing this, other predictors have a better chance to contribute to the splits. This process "decorrelates" the trees, making their average less variable and more reliable.

The main difference between bagging and random forests lies in the choice of the predictor subset size. If we use all predictors, it's equivalent to bagging. However, when using the square root of the total predictors, random forests show a reduction in the test error.

Using a small subset of predictors is particularly helpful when we have many correlated predictors.

Just like bagging, random forests don't overfit if we increase the number of trees (denoted as B). Therefore, in practice, a sufficiently large value of B is used until the error rate stabilizes.

In the process of constructing a random forest, multiple decision trees are created. Each tree is generated independently, and for each tree, a random vector Θ_k is generated. The generation of Θ_k is unrelated to the previously generated random vectors $\Theta_1, ..., \Theta_{k-1}$, but they are all drawn from the same distribution.

To grow a tree, the training set and the corresponding Θ_k are used. This combination results in a classifier h($\mathbf{x}$, Θ_k), where $\mathbf{x}$ represents an input vector. In bagging, for example, Θ is generated by counting the number of darts that randomly fall into N boxes, with N being the number of examples in the training set. In random split selection, the content of Θ depends on its usage in constructing the tree.

Once a large number of trees are generated, they collectively contribute to the final classification decision. Each tree casts a vote, and the class with the highest number of votes is considered the most popular. This aggregation of trees is what we refer to as a random forest.

A random forest is a classifier consisting of a collection of tree-structured classifiers $\{h(\mathbf{x},\Theta_k), k=1,\ldots\}$ where the $\{\Theta_k\}$ are independent identically distributed random vectors and each tree casts a unit vote for the most popular class at input $\mathbf{x}$.

We are going to use diabetes dataset gathered among the Pima Indians by the National Institute of Diabetes and Digestive and Kidney Diseases and compare the result of decision tree, bagged and random forest.

```r
misclassification_rate_vec <- rep(0,100)
misclassification_rate_vec_rf <- rep(0,100)
misclassification_rate_vec_bagged<- rep(0,100)

E_sim <- read.table('diabetes.csv',sep=',',header = TRUE)
E_sim$Outcome<- as.factor(E_sim$Outcome)

head(E_sim)
```

```
##   Pregnancies Glucose BloodPressure SkinThickness Insulin  BMI
## 1           6     148            72            35       0 33.6
## 2           1      85            66            29       0 26.6
## 3           8     183            64             0       0 23.3
## 4           1      89            66            23      94 28.1
## 5           0     137            40            35     168 43.1
## 6           5     116            74             0       0 25.6
##   DiabetesPedigreeFunction Age Outcome
## 1                    0.627  50       1
## 2                    0.351  31       0
## 3                    0.672  32       1
## 4                    0.167  21       0
## 5                    2.288  33       1
## 6                    0.201  30       0
```

The dataset consists of 768 cases, 8 variables and two classes. The variables are medical measurements on the patient plus age and pregnancy information. The classes are: tested positive for diabetes (268) or negative (500).

In the simulated data, Breiman uses 300 samples as learning set from the 1800 samples generated. However we are going to use the raw dataset and split the dataset into 2 - a learning set, L, (10% of the data) with 76 rows and a test set, T, (90% of the data) with 692 rows. Next we will construct the classification tree model using train() function with 10-fold cross-validation. A bagged decision tree is created in parallel using the function randomForest() and setting the number of trees as 25 and mtry (the number of variables used) as 8. Last we create a Random Forest model using the same function but here we mtry as 3 ($\sqrt{8}$). So the best 3 variables among the 8 will be chosen.

The three models are made to predict the result using the test dataset T and the miscalculation rate is calculated. The whole process is iterated 100 times for different combination of learning and test data and the mean error is calculated.

```r
for (i in 1:100){
  index <- createDataPartition(E_sim$Outcome, p = 0.7, list = FALSE)

  # Divide the data into test set T and learning set L
  T <- E_sim[-index, ]
  L <- E_sim[index, ]

  # Train the classification tree model with 10-fold cross-validation
  modeldt <- train(
    Outcome ~ .,
    data = L,
    method = "rpart",
    trControl = trainControl(method = "cv", number = 10)
  )

  rf_model <- randomForest(Outcome ~ ., data = L,mtry = 3, importance=TRUE)
  bagged_tree<- randomForest(Outcome ~ ., data = L,mtry = 8, ntree = 25)



  # Predict the class labels for the test set using the trained model
  predictions <- predict(modeldt, newdata = T)
  # Calculate misclassification rate
  misclassification_rate <- mean(predictions != T$Outcome)
  misclassification_rate_vec[i] <- misclassification_rate
  predictions_rf <- predict(rf_model, newdata = T)
  # Calculate misclassification rate
  misclassification_rate_rf <- mean(predictions_rf != T$Outcome)
  misclassification_rate_vec_rf[i] <- misclassification_rate_rf

  predictions_bagged <- predict(bagged_tree, newdata = T)
  # Calculate misclassification rate
  misclassification_rate_bagged <- mean(predictions_bagged != T$Outcome)

  misclassification_rate_vec_bagged[i] <- misclassification_rate_bagged

}
```

```
mis<- mean(misclassification_rate_vec)
mis_rf <- mean(misclassification_rate_vec_rf)
mis_bagg <- mean(misclassification_rate_vec_bagged)
err_data <- data.frame(Model = c("Decision Tree", "Bagging Predictor", "Random Forest"),
                       Error = c(mis,mis_bagg,mis_rf))
err_data
```

```
##               Model     Error
## 1     Decision Tree 0.2588696
## 2 Bagging Predictor 0.2493913
## 3     Random Forest 0.2386087
```

From the table above we can see that the Error is maximum for Decision Tree and least for Random Forest. Bagged Predictor model performs better than Decision Tree ie it can reduce the variance of an unstable dataset.

Larger Data Sets

Bagging Regression Trees

WHY BAGGING WORKS

Numeric Prediction

Instability is important in bootstrap aggregating because it helps improve the performance of the ensemble model by reducing variance and increasing accuracy. Bagging is a technique that combines multiple models (often decision trees) trained on different bootstrap samples of the dataset to make predictions. The key idea behind bagging is to introduce diversity among the base models by using different subsets of the training data. This diversity is achieved through the bootstrap sampling process, where each base model is trained on a randomly selected subset of the original data, allowing for variations in the training sets. If the base models used in bagging are stable, meaning they produce similar predictions when trained on slightly different datasets, the ensemble model's performance may not significantly improve. This is because the models will produce similar predictions and will not provide enough diversity to effectively reduce variance. On the other hand, if the base models are unstable, meaning they are sensitive to small changes in the training data, the ensemble model will benefit from the aggregation of these diverse models. The variations among the models can help to capture different aspects of the data and lead to improved generalization and prediction accuracy. Therefore, instability, or the ability of the base models to exhibit different behaviors with different training data, is important in bagging as it contributes to increased diversity and better performance of the ensemble model.

Let each (y,$\mathbf{x}$) case in L be independently drawn from the probability distribution P. Suppose y is numerical and $\Phi(\mathbf{x}, L)$ the predictor. Then the aggregated predictor is the average over L of $\Phi(\mathbf{x}, L)$, i.e. $\Phi_A(\mathbf{x}) = E_L[\Phi(\mathbf{x}, L)]$. Take x to be a fixed input value and y an output value. Applying the inequality $E(Z^2) \geq (E(Z))^2$ then $E_L[(y - \Phi(\mathbf{x}, L)^2] \geq (y - \Phi_A(x))^2$. Integrating both sides over the joint y,$\mathbf{x}$ output-input distribution, we get that the mean_squared error of $\Phi_A(\mathbf{x})$ is lower than the mean-squared error averaged over L of $\Phi(\mathbf{x}, L)$. How much lower depends on how unequal the two sides of $[E_L\delta(\mathbf{x}, L)]^2 \leq E_L\delta^2(\mathbf{x}, L)$ are. The effect of instability is clear. If $\delta(\mathbf{x}, L)$ does not change too much with replicate L the two sides will be nearly equal, and aggregation with not help. The more highly variable the $\delta(\mathbf{x}, L)$ are, the more improvement aggregation may produce. But δ_A always improves on δ. Now Φ_A depends not only on $\mathbf{x}$ but also the underlying probability distribution P from which L is drawn, i.e. Φ_A= $\Phi_A(\mathbf{x}, P)$. But the bagged estimate is not $\delta_A(\mathbf{x}, P)$ but rather $\delta_B(X) = \delta_A(\mathbf{x}, P_L)$ where P_L is the distribution that concentrates 1/N at each point $(y_n, \mathbf{x}_n) \in L$. Then if the procedure in unstable, δ_B can give improvement through aggregation. On the other side, if the procedure is stable, then δ_B will not be as accurate for data drawn from P as $\delta_A(\mathbf{x}, P)$. There is a cross-over point between instability and stability at which δ_B stops improving on $\delta_A(\mathbf{x}, P)$ and does worse.

Classification

In classification, a predictor $\delta(\mathbf{x}, L)$ predicts a class label j $\in$ {1,. . . ,J}. Denote Q(J|$\mathbf{x}$)= P($\Phi(\mathbf{x}, L) = j$). The interpretation of Q(J|$\mathbf{x}$) is this: over many independent replicates of the learning set L, Φ predicts class label j at input $\mathbf{x}$ with relative frequency Q(J|$\mathbf{x}$). Let $P(J|\mathbf{x})$ be the probability that input $\mathbf{x}$ generates class j. Then the probability that the predictors classifies the generated state at $\mathbf{x}$ correctly is $\sum_j Q(J|\mathbf{x})P(J|\mathbf{x})$. The overall probability of correct classification is r= $\int[Q(J|\mathbf{x})P(J|\mathbf{x})]P_x(d_x)$ where $P_x(d_x)$ is the $\mathbf{x}$ probability distribution. Note that for any $Q(j|\mathbf{x})$, $\sum_j Q(J|\mathbf{x})P(J|\mathbf{x}) \leq \max\_jP(J|\mathbf{x})$ with equality only if

$$\begin{cases} 1 & if \quad P(j|\mathbf{x}) = max_i P(i|\mathbf{x})) \\ O & else \end{cases}$$

The predictor $\Phi^*(\mathbf{x})$= $argmax_j P(J|\mathbf{x})$ leads to the above expression for $Q(J|\mathbf{x})$ and gives the highest attainable correct classification rate: r^*=$\int max_j P(j|\mathbf{x})P_X()$. Call Φ order-correct at the input $\mathbf{x}$ if $argmax_j Q(j|\mathbf{x}) = argmax_j P(j|\mathbf{x})$. This means that if input $\mathbf{x}$ results in class j more often than any other class, then Φ also predicts class j at $\mathbf{x}$ more often than any other class. The aggregated predictor is: $\Phi_A(\mathbf{x}) = argmax_j Q(j|(x))$. For the aggregated predictor the probability of correct classification at $\mathbf{x}$ is $\sum_j I(argmax_i Q(i|\mathbf{x}) = j)P(j|\mathbf{x})$. If Φ is order_correct at $\mathbf{x}$, then the previous formula equals $max_j P(j|\mathbf{x})$. Letting C be the set of all inputs $\mathbf{x}$ at which Φ is order-correct, we get for the correct classification probability of Φ_A the expression:

$$r_A = \int_{\mathbf{x} \in \mathbf{C}} max_j P(J|\mathbf{x})P_x(d\mathbf{x}) + \int_{\mathbf{x} \in \mathbf{C}'} [\sum_j I(\Phi_A(\mathbf{x}) = j)P(j|\mathbf{x})]P_x(\mathbf{x}))]$$

Even if Φ is order-correct at $\mathbf{x}$ its correct classification rate can be far from optimal. But Φ_A is optimal. If a predictor is good in the sense that it is order-correct for most inputs $\mathbf{x}$, then aggregation can transform it into a nearly optimal predictor. On the other hand, unlike the numerical prediction situation, poor predictors can be transformed into worse ones. The same behavior regarding stability holds. Bagging unstable classifiers usually improves them. Bagging stable classifiers is not a good idea.

Using the learning set as a test set

In bagging trees, the training set L_B is generated by sampling from the distribution P_L. Using L_B a large tree T is constructed. The standard CART methodology find the sequence of minimum cost pruned subtrees of T. The best pruned subtree in this sequence is selected either using cross-validation or a test set. The idea of a test is that it is formed by independently sampling from the same underlying distribution that gave rise to the learning set. In the present context, a test set is formed by independently sampling from P_L, i.e. we can get a test set by sampling with replacement from the original learning set L. Infact, consider sampling with replacement a large number of times from L, each case $(y_n, \mathbf{x_n})$ will be selected about the same number of time as any other case. Thus, using a very large test set sampled from P_L is equivalent to just using L as a test set.

A LINEAR REGRESSION ILLUSTRATION

Forward variable selection

With data of the from $L = (y_n, \mathbf{x_n}), n = 1...N$ where $\mathbf{x} = (x_1, ..., x_M)$ consists of M predictor variables, a popular prediction method consists of forming predictors $\delta_1(\mathbf{x})...\delta_M(\mathbf{x})$ where $\delta_m(\mathbf{x})$ is linear in $\mathbf{x}$ and depends on only m of the M x-variables. Then one of the $\delta_m(\mathbf{x})$ is chosen as the designated predictor. A common method for constructing the $\delta_m(\mathbf{x})$, and one that is used in our simulation, is forward variable entry. If the variables used in $\delta_k(\mathbf{x})$ are $(x_{1k}, ..., x_{mk})$ then for each m $\notin m_1, ...m_K$ form the linear regression of y on $(x\_m1, ..., x\_mk + 1, x_m)$, compute the RSS(m) and take x_{mk+1} such that m_{k+1} minimizes RSS(m) and $\delta_k + 1(\mathbf{x})$ the linear regression based on $(x_{m1}, ..., x_{mk+1})$. What is clear about this procedure is that it is an unstable procedures. The variables are competing for inclusion in the δ_m and small changes in the data can cause large changes in the δ_m. Forward variable selection is considered an unstable procedure due to the following reasons:

1. Dependency on the order of variable selection: in forward variable selection, the procedure starts with an empty model and adds one variable at a time based on certain selection criteria. However, the initial selection of variables can significantly impact the final result. If the initial set of selected variables differs, the progressive selection of variables can lead to different models. Therefore, the selection of initial variables can cause instability in the results.

2. Sensitivity to correlated variables: forward variable selection may be influenced by correlated variables. When adding a variable to the model, it may appear to provide significant informative value, but it could be strongly correlated with other variables already present in the model. This correlation can lead to an overestimation of the importance of the added variable and result in selecting a suboptimal model.

3. Impact of small data variations: forward variable selection can be sensitive to small variations in the input data. Even a slight modification of the input data can lead to a completely different variable selection or a different model. This instability can make it challenging to reproduce the results.

To mitigate the potential instability of forward variable selection, techniques such as cross-validation or bootstrap can be used to assess the stability of selected variables and the robustness of the resulting model. These techniques allow for a more accurate estimation of model performance and more reliable identification of the most relevant variables.

Simulation structure

The simulated data used in this section are drawn from the model y= $\sum_m \beta_m x_m + \epsilon$ where ϵ is N(0,1). The number of variables M=30 and the sample size is 60. The subset selection is nearly optimal is there are only a few large non-zero β_m , and its performance is poor if there are many small but non-zero β_m. Subset selection methods can indeed perform well when there are only a few large non-zero coefficients. This is why subset selection can be effective in such cases:

1. Focus on important predictors: subset selection methods aim to identify a subset of predictors that are most relevant for predicting the response variable. When there are only a few predictors with large non-zero coefficients, subset selection can successfully identify and include those important predictors in the model.

2. Improved model interpretability: by selecting a subset of predictors with large non-zero coefficients, subset selection can lead to a more interpretable model. The resulting model will focus on the most influential predictors, providing insights into the relationship between these predictors and the response variable.

3. Reduction in model complexity: by selecting a smaller subset of predictors, subset selection reduces the complexity of the model compared to including all predictors. This reduction in complexity can improve model generalization, making the model more robust and less prone to overfitting.

It's important to note that subset selection methods, like forward selection or backward elimination, are not guaranteed to find the exact optimal model in all cases. They are heuristic approaches that explore different combinations of predictors. The optimality of the selected model depends on factors such as the quality of the selection criteria and the sample size. In practice, it's always recommended to evaluate the performance of the selected model using cross-validation or other validation techniques like bootstrap. This helps to assess the model's predictive accuracy and determine if additional refinement or regularization techniques are needed. In summary, when there are only a few large non-zero coefficients, subset selection methods can be effective in identifying the most important predictors and constructing a simpler, interpretable model. However, it's crucial to validate the model and consider other factors to ensure its performance and reliability.

Three sets of coefficients are used. In the first set of coefficients there are only three non-zero β_m. In the second set of coefficients there are 15 non-zero β_m. In the third set there are 27 non-zero β_m, all relatively small. For each set of coefficients, the following procedure was replicated 250 times: 1. Data L=$L = (y_n, \mathbf{x_n}), n = 1...60$ was drawn from the model y=$\sum_m \beta_m x_m + \epsilon$. 2. Forward entry variables was done using L to get the predictors $\delta_1(\mathbf{x})...\delta_M(\mathbf{x})$. The mean_squared prediction error of each of these was computed giving $e_1...e_M$. 3. Fifty bootstrap replicates L_B of L were generated. For each of these, forward step-wise regression was applied to construct predictors $\delta_1(\mathbf{x}, L_B)...\delta_M(\mathbf{x}, L_B)$. These were averaged over the L_b to give the bagged sequence $\delta_1^B(\mathbf{X})...\delta_M^B(\mathbf{X})$. The prediction errors $e_1^B...e_M^B$ for this sequence were computed.

These computed mean-squared-errors were averaged over the 250 repetitions to give the sequences $\bar{e}_m^S, \bar{e}_m^B$. For each set of coefficients, these two sequences are plotted vs m in Figure1 a,b,c.

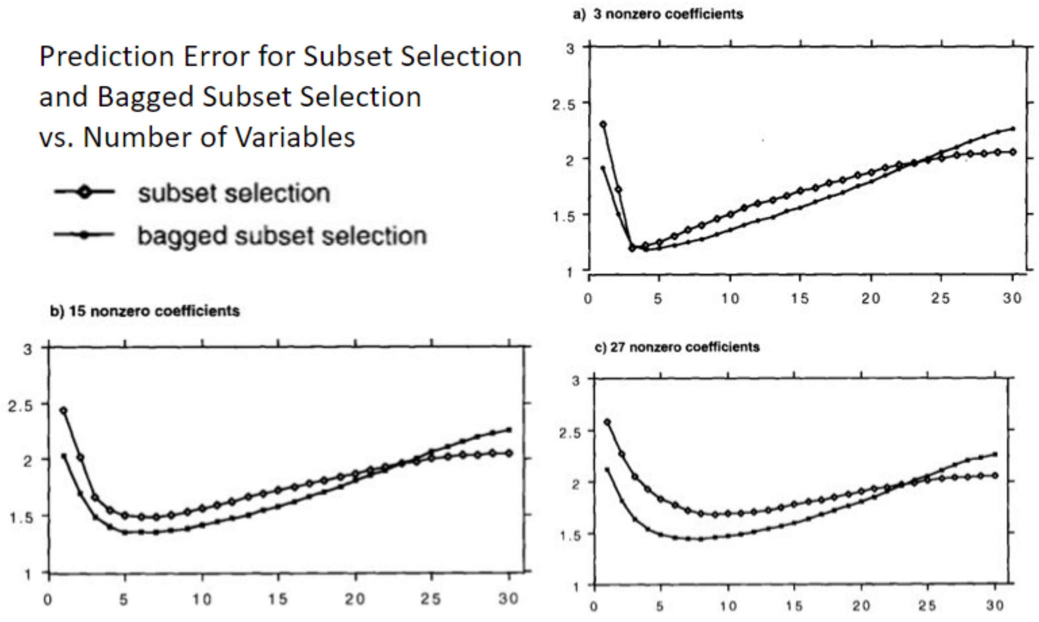


Figure 1: Figure 1. Prediction error for subset selection and bagged subset selection

Discussion of simulation results

Looking at Figures1 a,b,c an obvious result is that the most accurate bagged predictor is at least as good as the most accurate subset predictor. In the first set of coefficients (with only three non-zero β_m), subset selection is nearly optimal and there is no improvement. In the second and third set of coefficients there is substantial improvement. *Bagging can improve only if the unbagged is not optimal.* Note that in all three graphs there is a point past which the bagged predictors have larger prediction error than the unbagged. The explenation is this: linear regression using all variables is a fairly stable procedure. The stability decreases as the number of variable used in the predictor decreases.*For a stable procedure $\delta_B = \delta_A(\mathbf{x}, P_L)$ is not as accurate as $\delta(\mathbf{x}, P)$.* As m increases, the instability increases and there is a cross-over point at which δ_m^B

becomes more accurate than δ_m.

CONCLUDING REMARKS

Bagging class probability estimates

Some classification methods estimate probabilities p(j|**x**) that an object with prediction vector **x** belongs to class j. Then the class corresponding to **x** is estimated as $argmax_j p(j|\mathbf{x})$. For such methods, a natural competitor to bagging by voting is to average the p(j|**x**) over all bootstrap replications, getting $p_B(j|\mathbf{x})$ and then use the estimated class $argmax_j p_B(j|\mathbf{x})$. In some applications, estimates of class probabilities are require instead of, or along with, the classifications. The evidence so far indicates that bagged estimates are likely to be more accurate than the single estimates. To verify this, it would be necessary to compare both estimates with the true values p*(j|**x**) over the **x** in the test set. For real data the true values are unknown but they can be computed for the simulated waveform data. Using the waveform data, a simulation with learning and test sets both of size 300 and 25 bootstrap replications has been done. In each iteration, the average of |p(j|**x**)-p*(j|**x**)| and |p_B(j|**x**)-p*(j|**x**)| was computed. This was repeated 50 times and the results averaged. The single tree estimated had an error of .189. The error of the bagged estimates was .124, a decrease of 34%.

How many bootstrap replicate are enough?

In the paper's experiment, 50 bootstrap replicates was used for classification and 25 for regression. To give some ideas of what the results are as connected with the number of bootstrap replicates we ran the waveform data using 10,25,50 and 100 replicates using the same simulation scheme. The results appear in Table 10. The unbagged rate is 29.1, so its clear that we are getting most of the improvement using only 10 bootstrap replicates.

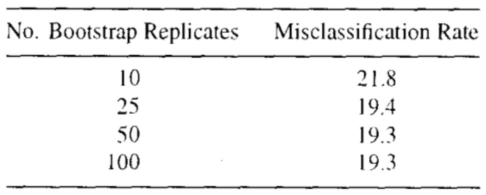


Table 10. Bagged Misclassification Rates (%)

No. Bootstrap Replicates	Misclassification Rate
10	21.8
25	19.4
50	19.3
100	19.3

How big should the bootstrap learning set be?

In the paper's simulations the bootstrap replicates L^B were used of the same size as the initial learning set L. While a bootstrap replicate may have 2,3,... duplicates of a given instance, it also leaves out about .37 of the instances. A reader of the technical report on which the paper is based remarked that this was an appreciable loss of data, and that accuracy might improve if a larger bootstrap set was used. Anyway with a bootstrap learning sets twice the size of L there was no improvement in accuracy.

Conclusions

Bootstrap Aggregating can be a valuable approach to improve the performance of a predictive model, especially when the initial model is not very accurate or reliable. Bagging is a technique that involves creating multiple bootstrap samples from the original data and fitting a separate model to each sample. The models are then combined or aggregated to make predictions.Implementing bagging is relatively straightforward, as it requires adding a loop to select bootstrap samples and sending them to the modeling procedure, as well as an aggregation step at the end. However, when using bagging with decision trees as the base model, one may lose the simplicity and interpretability of the individual trees. Decision trees typically have a clear and interpretable structure that allows for easy understanding of the underlying rules. In the bagging process, where multiple trees are combined, the resulting ensemble model may not have a straightforward and interpretable structure. Despite this loss of interpretability, the advantage of using bagging with decision trees is increased accuracy. By combining multiple trees through bagging, the variance is reduced, leading to improved predictive performance.

In summary, bagging is a valuable technique for improving the accuracy of a predictive model, even at the cost of losing the simplicity and interpretability of individual trees and it allows for increased accuracy through the combination of multiple models.

Comparison

Bootstrapping is random sampling with replacement from the available training data. Bagging (= bootstrap aggregation) is performing it many times and training an estimator for each bootstrapped dataset.

When you just/only do bootstrap without an aggregation at the end you only use the standard deviation of the bootstraps replications together, than you call it bootstrap.

If you do the same, but at the end you take an aggregation of voting or anything else, than you call it bagging. They are both ensemble technique. One of the technique in bagging is called as Random Forest. In Random Forest you basically use multiple decision trees.

References

Breiman L. (1996). *Bagging Predictors*, Statistics Department, University of California, Berkeley.

Breiman L. (2001). *Random forest*, Statistics Department, University of California, Berkeley.

Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani (2021). *An Introduction to Statistical Learning with Applications in R*, Second Edition.

Tae-Hwy Lee, Aman Ullah and Ran Wang. *Bootstrap Aggregating and Random Forest.*

Andreas Buja, Werner Stuetzle.*Observation on Bagging.* University of Pennsylvania and University of Washington.

Peter Bühlmann, Bin Yu (2002).*Analyzing bagging*, ETH Zürich and University of California, Berkely.