Discovering Simple Voting Ensembles with *k*-Fold Cross-Validation

Timothy Mitchell **TMITCHELL8@LUC.EDU**

Department of Mathematics and Statistics, Loyola University of Chicago, Chicago, IL

Abstract

Ensembles are systems of models that work together to make predictions. In a simple voting ensemble, predictions are decided by majority vote, with votes cast by two or more base learners. While ensembles are known to have good performance, preparing a voting ensemble can be challenging; often, it is not clear which base learners help or hurt the final model. This paper proposes a supervised learning method for evaluating simple voting ensembles given a population of base learners. First, prediction vectors are computed for each base learner during repeated *k*-fold cross-validation. Second, prediction vectors are aggregated in all possible combinations to systematically evaluate a huge number of voting ensembles, with very little added cost to computational time. Finally, the method calculates the expected loss during repeated *k*-fold cross-validation for each base learner and voting ensemble. I apply this method to the Wisconsin diagnostic breast cancer data set and show that multiple voting ensembles outperform the other models in the system. In addition, the method is automatic. To find the best ensemble, the only inputs required are the data set and a chosen population of base learners. Advantages and disadvantages of the method are discussed.

Keywords: voting ensembles, classification, machine learning, autoML, repeated *k*-fold cross-validation

1. Introduction

Ensembles are systems of models working together. Tukey proposed one of the first examples of an ensemble when he combined two linear regression models to reduce the error (1977). Since then, ensembles have been applied successfully to a wide range of classification and regression problems, with numerous methods described. Wolpert (1992) introduced stacking ensembles, which train on the original data as well as the model's residuals in an effort to estimate, and ultimately undo, the bias. Breiman (1996) introduced bagging ensembles, which add stability to classifiers that otherwise suffer from overfitting and undesirable variance. Friedman (1999a) authored a number of groundbreaking papers on boosting ensembles, which optimize a differentiable loss function in the function space. Aleryani et al. (2020) studied ensembles in the context of missing data, and found that models trained on multiple imputations delivered better performance than models trained on single imputations. While theoretical studies of ensembles are less common, van der Laan et al. (2007) provided a proof that certain systems of models in the class of weighted voting ensembles perform asymptotically as well as the best model in the system.

Error Correction

To see how ensembles improve predictions, suppose $B_1, \dots, B_n$ follow a Bernouli distribution with parameter p. That is,

$$\mathbb{P}[B_i = 1] = \mathbb{P}[success] = p$$

$$\mathbb{P}[B_i = 0] = \mathbb{P}[failure] = 1 - p$$

A sequence of Bernoulli random variables is a binary vector $\boldsymbol{B} \in \{0, 1\}^n$.

What if we have N equally sized vectors $\boldsymbol{B_1}, \dots, \boldsymbol{B_N}$ and the goal is to minimize $\mathbb{P}[failure]$? One strategy is to average the vectors together. Since the expected value of a Bernoulli random variable is p, it follows that as $N \Rightarrow \infty$, the mean value of $B_i \Rightarrow p$.

After averaging N equally sized vectors $\boldsymbol{B_1}, \dots, \boldsymbol{B_N}$, we can apply the rule:

$$B_i = \begin{cases} 1, & B_i < p \\ 0 & otherwise \end{cases}$$

then $\mathbb{P}[success] \Rightarrow 1$ by the law of large numbers i.f.f. $B_i \sim \text{Bernoulli}(p)$ and $p > 0.5$ for all $\boldsymbol{B_1}, \dots, \boldsymbol{B_N}$,.

The previous result is also illustrated by the example of error-correcting code. Imagine a radio operator needs to send a binary signal to an ally: {`1 0 1 0 1 1`}.

Unfortunately, the operator has faulty equipment, so there is probability $1 - p$ that each of the digits is sent in error (that is, a `1` becomes a `0` or a `0` becomes a `1`).

How might the operator guarantee the correct signal is sent? One solution is to send the signal in batches, and decode the signal by computing the majority vote. Below, the sequence computed by majority vote eliminates the error.

```
Signal 1: 1 1 1 0 1 1
Signal 2: 1 0 1 0 0 1
Signal 3: 1 0 1 1 1 1
Signal 4: 1 0 0 0 1 1
Signal 5: 0 0 1 0 0 1
---------------------
Majority: 1 0 1 0 1 1
```

Voting Ensembles

Voting ensembles operate using the same principle.

```
 Model 1: 1 1 1 0 1 1
 Model 2: 1 0 1 0 0 1
 Model 3: 1 0 1 1 1 1
---------------------
Ensemble: 1 0 1 0 1 1
```

Here the sequences encode binary prediction vectors. If the predictions for each model are independent vectors of Bernoulli random variables with $p > 0.5$ and we tally the majority vote for many such models, it follows that the error rate goes to zero.

In practice, it is unreasonable to assume that model predictions are totally independent. Instead, observations that are difficult to classify for one model may be difficult to classify for another. Intuitively, if we aggregate several models that all make the same mistake, there is no improvement.

```
 Model 1: 1 1 1 0 1 1
 Model 2: 1 1 1 0 1 1
 Model 3: 1 1 1 0 1 1
---------------------
Ensemble: 1 1 1 0 1 1
```

It is also possible for a voting ensemble to do *worse* than its base learners. To illustrate, we propose three models each with an error rate of ⅓ or 33%, while the final ensemble has an error rate of ½ or 50%. It seems to be the worst-case scenario when models have a high error rate *and* the errors are correlated.

```
 Model 1: 1 1 1 0 0 1
 Model 2: 0 1 1 0 1 1
 Model 3: 0 0 1 0 0 1
---------------------
Ensemble: 0 1 1 0 0 1
```

Apparently, two conditions must hold for voting ensembles to succeed. First, models need to make good predictions individually; second, the errors need to be adequately uncorrelated.

Different strategies to decorrelate models have been proposed and tested. In a random forest (2001), trees are grown using bootstrapped samples of the data, which introduces variation in the trees. In the Guerts et al. model "extremely randomized trees" (2006), the cut-points in a random forest are fully randomized (but each tree is grown using all available data). In stacking (1992), new models are trained using the residuals of the old models, so that the models are trained on different data sets. In boosting (Friedman, 1999a), the model weights are revised at each iteration, which adds diversity to the base learners. Stochastic gradient descent goes a step further, training base learners on subsamples of the data (Friedman, 1999b). In multiple imputation ensembles (2020), diversity comes from the set of multiple imputations. Ensembles of neural networks enjoy many advantages; for example, weights are initialized randomly at each run (2002), and dropout (2014) can thin a neural network by randomly eliminating neurons. The most successful ensembles seem to involve diversity, usually in the training data or the parameters.

When the strategy is to employ several state-of-the-art models at once, diversity comes from the different learning algorithms.

Ensemble Diversity

Despite the prevailing wisdom that ensembles benefit from diversity, attempts to understand and quantify this "diversity" have not been met with a consensus.

On the one hand, Liu et al. (1999) proposed "negative correlation learning" to induce diversity among base learners, and Yu et al. (2011) proposed a "diversity regularized machine"; both groups of authors claimed experimental support for their methods.

On the other hand, Kuncheta and Whitaker measured diversity in binary prediction vectors using 10 statistics ("Measures of Diversity in Classifier Ensembles and Their Relationship with Ensemble Accuracy," 2003). "Although there are some special cases," they wrote, "our results raise some doubts about the usefulness of diversity measures in building classifier ensembles in real-life pattern recognition problems." Bian and Chen (2019) revisited the problem more recently, writing "the relationship between diversity and generalization [...] is not entirely understood and remains an open research issue."

Ensemble Performance

Despite some holes in the theoretical understanding of ensembles, their accuracy is not disputed. It is widely accepted that a carefully tuned ensemble generalizes better than one single model; see for example Wu et al. (2010), Rokach (2010), Herrera et al. (2016), and Gu et al. (2015). A useful case study is the Otto Product Classification Challenge (2015), a Kaggle competition in which the winning solution was an ensemble of 37 base models, including neural networks, *k*-nearest neighbors, and random forests. Since 2015, large-scale Kaggle competitions almost always feature ensembles at the top of the leaderboards. In 2017, the winners of the Statoil/C-CORE Iceberg Classifier Challenge used an ensemble of more than 100 neural networks.

2. Methods

My goal was to fit an ensemble classifier that could outperform the models in its network. In summary, I trained L learners on the data. Next, I tested each model using 10 cycles of 5-fold cross-validation and saved the out-of-sample predictions at each iteration of 5-fold cross-validation. Finally, taking the L prediction vectors generated at each fold, I calculated expected loss for each of the L models and expected loss for each of the $2^L - 1$ possible voting ensembles. I review in turn the data, the choice of loss function, my justification for using repeated *k*-fold cross-validation, and the code by which I evaluated voting ensembles.

Data

I evaluated ensembles using the Wisconsin diagnostic breast cancer data set (1993) from the UCI machine learning repository. The data set contains 30 numeric features describing tumors from 569 cancer patients. Each tumor has a single binary outcome label, benign ($n = 357$) and malignant ($n = 212$). The purpose of data collection was to provide clinicians with a set of features to aid in clinical diagnosis (Street, Wolberg, & Mangasarian, 1993). The complete list of features is given in **Supplementary Table 1**.

Loss Function

For a binary classification problem, the outcome Y has 2 mutually exclusive class labels. Denote X_i as the set of observations in the training data, $i = 1, \ldots, n$, and F as the *p*-dimensional set of covariates. We denote $\Psi(F) = \mathbb{E}[Y \mid F]$ as the estimate of the response. A convenient expression for $\Psi(F)$ is the function:

$$\Psi(F) = argmin\, \mathbb{E}[L(X, \Psi(F))]$$

Loss is often a minimizer of binary crossentropy or mean square error. I chose as my loss function Cohen's kappa (1960), a coefficient of agreement appropriate for unbalanced classes. Unlike accuracy, which might be misleading when classes are imbalanced, κ considers the marginal distributions of each of the classes.

$$\kappa = \frac{observed\,agreement - expected\,agreement}{1 - expected\,agreement}$$

Classifiers

I considered 11 models in my simulations. Models and their `R` libraries included: logistic regression (`stats`), *k*-nearest neighbors (`class`), conditional inference trees (`partykit`), support vector machines (`e1071`), extremely randomized trees (`ranger`), elastic net (`glmnet`), random forest (`ranger`), neural networks (`neuralnet`), linear discriminant analysis (`MASS`), quadratic discriminant analysis (`MASS`), and XGBoost (`xgboost`).

For some models, I did not need to do much tuning, but for others, such as XGBoost, parameter tuning was an intensive process. For XGBoost I defined a grid of 2400 combinations of parameters, such as tree depth, learning rate, gamma, sample fraction, and the number of features randomly sampled at each node. A list of model parameters is given in **Supplementary Table 2**.

In general, I tuned models using grid searches. To do this, I iterated through each row of a parameter grid, training the model at each iteration. I estimated the generalization error using 5-fold cross-validation and averaged the value of κ across 6 to 10 cross-validation cycles.

I selected the set of optimal parameters θ based on the model $\Psi(F, \theta)$ that minimized the expected loss for the combinations tested:

$$\Psi(F, \theta) = argmin\, \mathbb{E}[1 - \kappa(X, \Psi(F, \theta))]$$

Parallel Programming

I found that parallel programming reduced the runtime for grid searches. For example, I tested 465 combinations of parameters for extremely randomized trees, growing 650 trees at each run. With a nested for loop, 10 cycles of 5-fold cross-validation required 3414.5 seconds. Using `mcmapply` to parallelize a key vector operation reduced the time to 3047.3 seconds, a 12% reduction. Using `foreach` to parallelize the outer loop and retaining `mcmapply` in the inner loop reduced the time even further to 3023.5 seconds.

Repeated k-Fold Cross-Validation

Like any estimator, k-fold cross-validation has some variance. If we apply k-fold cross-validation twice for the same data, but with a different random partioning of the data into k folds, then these two procedures can give quite different results. *Repeated* cross-validation attempts to rectify this by averaging together the error estimates from multiple k-fold cross-validation cycles. In our case, additional cycles of k-fold cross-validation might improve the precision of the point estimate of κ, which helps determine the best voting ensemble.

Besides the variance, k-fold cross-validation has some bias. This has been pointed out by Hastie et al. (2011) and others. The error estimates given by the procedure are often pessimistic. By repeatedly training the model on a subset of the data, k-fold cross-validation deprives the model of valuable learning experience that should, in theory, improve the model trained on all the data.

The relationship between bias and variance depends on the chosen value of k. Kohavi (1995) studied the problem and determined that larger k leads to smaller bias but higher variance.

Vanwinckelen and Blockeel (2013) offer a well-argued criticism of repeated k-fold cross-validation. According to the authors, repeated cross-validation estimates the mean $\bar{x}_k$ of the loss computed for all possible k-fold cross-validations over the sample S. This is not the same, they argue, as the true generalization error α, since k-fold cross-validation is empirically biased. They conclude, "the mean [$\bar{x}_k$] converges to another value than any of the values one might really be interested in estimating."

I argue that, because $\bar{x}_k$ seems to be correlated with α, it is a meaningful estimator.

Responding to Vanwinckelen and Blockeel

In their experiments, authors applied cross-validation to nine different data sets and computed the mean cross-validation error $\bar{x}_k$ for single-cycle and repeated cross-validation. Using a holdout set, they estimated the true generalization error α and compared it to $\bar{x}_k$. Authors used two different models, C4.5 and support vector machines. Authors criticized repeated k-fold cross-validation on the basis it did not reduce the bias ($|\bar{x}_k - \alpha|$) in the error estimate.

Unfortunately, the authors failed to consider that $\bar{x}_k$ might still be a useful approximation for α. I calculated the Pearson correlation coeffient ρ between $\bar{x}_k$ and α using the authors' own data (see Vanwinckelen and Blockeel, 2013, Table 2). I wondered: do the error estimates $\bar{x}_k$ computed by cross-validation have a stronger correlation with the true generalization error α if we average together several cycles? Evidence seems to support this idea. The correlation ρ between $\bar{x}_k$ and α computed across nine experiments was highest for repeated cross-validation. The relationship was stronger for small sample sizes ($S = 100$) compared to large sample sizes ($S = 1000$).

This might be an indication that, while $\bar{x}_k$ is biased, it is still correlated with the true generalization error α. If that is true generally, then the use of repeated k-fold cross-validation to improve error estimates is justified, at least for small sample sizes. I am not convinced that repeated k-fold cross-validation should be abandoned simply because it is biased.

Table 1: Correlation Between the Cross-Validation Error and Generalization Error (9 Data Sets) for Different Models, Sample Sizes, and CV Cycles.

	CV	**10× CV**	**30× CV**
C4.5 model, S = 100	0.985	0.992	0.992
SVM model, S = 100	0.988	0.992	0.993
C4.5 model, S = 1000	0.997	0.998	0.997
SVM model, S = 1000	0.998	0.999	0.999

A final note about cross-validation is that it represents a kind of simulation. Essentially, when we don't have a validation set, we treat the *sample* as the population and *each set of k – 1 folds* as the sample. In this way we simulate the task of obtaining a random sample from a population and testing a model on unseen data. Repeated cross-validation can be seen as a Monte Carlo technique for minimizing $\mathbb{E}[1 - \kappa(X, \Psi(F))]$ for the set of all possible k-fold cross-validations on sample S.

Algorithm for Estimating the Expected Loss of the Base Learners

Estimating the expected loss during c cycles of k-fold cross-validation requires a nested for loop. Pseudocode is printed on the following page. The outer loop (Line 1) iterates through c cycles of k-fold cross-validation. When the loop is initialized, the data is randomly partitioned into k folds (Line 3). The inner loop (Line 5) performs k-fold cross-validation. The loss is evaluated k times for Model 1 (Line 11) and Model 2 (Line 14) and then saved to a unique matrix pre-allocated for each model. The row means of each of these matrices

(Line 20 and Line 21) give the expected loss for each cycle of *k*-fold cross-validation. The row means of these objects (Line 27 and Line 28) in turn give the expected loss $\mathbb{E}[1 - \kappa(X, \Psi(F))]$ for *c* cyles of *k*-fold cross-validation.

Algorithm 1

```
1.  for (c in 1:cycles) {
2.
3.    fold <- divide.into.folds(data)
4.
5.    for (k in 1:k){
6.
7.      train <- data[fold != k]
8.      test <- data[fold == k]
9.
10.     mod1 <- model(data = train)
11.     loss1[ , k] <- loss(predicted, test)
12.
13.     mod2 <- model(data = train)
14.     loss2[ , k] <- loss(predicted, test)
15.
16.     # model 3, 4, 5, etc.
17.
18.   }
19.
20.   expected_Loss1[ , c] <- rowMeans(loss1)
21.   expected_Loss2[ , c] <- rowMeans(loss2)
22.
23.   # model 3, 4, 5, etc.
24.
25. }
26.
27. mod1_accuracy <- rowMeans(expected_Loss1)
28. mod2_accuracy <- rowMeans(expected_Loss1)
29.
30. # model 3, 4, 5, 5, etc.
```

Algorithm for Computing the Expected Loss of the Base Learners and Voting Ensembles

Evaluating voting ensembles requires aggregating the predictions of the base learners in all possible combinations. Since the predictions of the base learners were evaluated in Algorithm 1, it is not necessary to fit each of the models again. We just need to export predicted and true valies in the course of Algorithm 1 and call them again in Algorithm 2. As before, we calculate the expected loss for *k* iterations of *c* cyles.

For *L* models, we have $2^L - 1$ meaningful combinations. For example, for 3 models A, B, and C, the meaningful combinations are {A}, {B}, {C}, {A, B}, {A, C}, {B, C}, and {A, B, C}. (It is possible one of the base learners has lower error, or it is possible one of the ensembles has lower error. Both possibilities are interesting.) We can also express this using binary notation: {`1 0 0`}, {`0 1 0`}, {`0 0 1`}, {`1 1 0`}, {`1 0 1`}, {`0 1 1`}, and {`1 1 1`}. The matrix of meaningful combinations is defined this way (Line 4) as a function of the number of base learners *L* (Line 1).

To avoid ties, voting ensembles with an even number of members were not considered (Line 9). This also cuts the number of calculations in half, as the number of combinations is reduced to about $(2^L - 1)/2$.

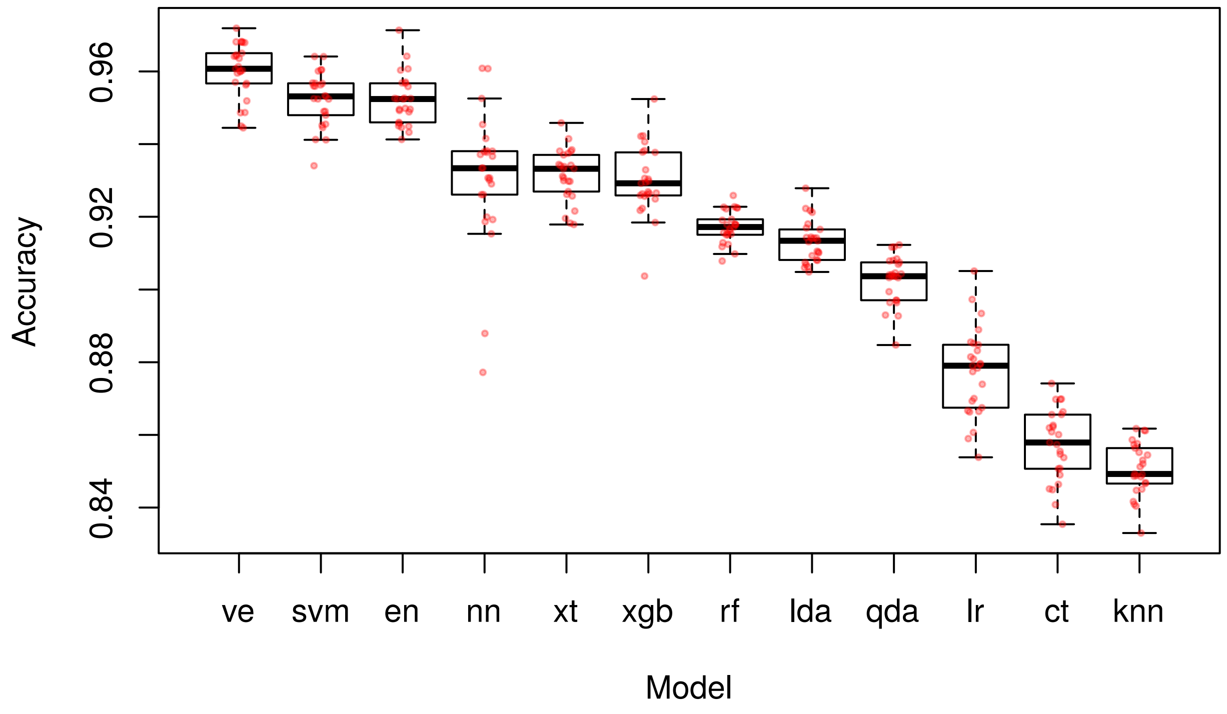
Finally, a triple-nested for loop (Line 15) computes the expected loss for each of *L* models for *c* cycles of *k*-fold cross-validation. The basic strategy is to multiply the matrix `voters` by the set of all prediction vectors. Some of the values in this matrix are `NA` (Line 12), so that when the matrix `voters` and prediction vectors are multiplied out, some of the voting ensembles will have only a few models, some will have many, and sometimes, a base learner is left alone. This operation was too complex to fully summarize with pseudocode, but it happens near Line 25.

As before, I saved the expected loss for each fold and each cycle (Line 31) and the row means of this object (Line 35) give the expected loss $\mathbb{E}[1 - \kappa(X, \Psi(F))]$ for *c* cyles of *k*-fold cross-validation.

Algorithm 2

```
1.  L <- number.of.models()
2.
3.  # 2^L - 1 voters
4.  voters <- do.call(expand.grid,
5.                    replicate(L,
6.                    list(0:1)))[-1, ]
7.
8.  # around (2^L - 1)/2 voters
9.  voters <- voters[apply(voters,
10.         1, function(x) sum(x) %% 2 != 0), ]
11.
12. voters[voters == 0] <- NA
13.
14. # number of cycles
15. for (z in 1:c){
16.
17.   # number of folds
18.   for (y in 1:k){
19.
20.     # number of voting ensembles
21.     for (x in 1:nrow(voters)){
22.
23.       # loss computed for each model
24.       # including voting ensembles
25.       loss[[x, y]] <- loss(. . .)
26.
27.     }
28.
29.   }
30.
31.   expected_Loss[ , z] <- rowMeans(loss)
32.
33. }
34.
35. accuracy <- rowMeans(expected_Loss)
```

Figure 1: Accuracy Estimated by 25 Cycles of 5-Fold Cross-Validation



<u>Figure 1</u>: Accuracy (Cohen's κ coefficient) for 12 models. Each red point represents average accuracy estimated by one cycle of 5-fold cross-validation. 25 cycles are shown. Standard quartiles are indicated for each boxplot. *Models: ve,* voting ensemble*; *svm*, support vector machine; *en*, elastic net; *nn*, neural network; *xt*, extremely randomized trees; *xgb,* XGBoost; *rf*, random forest; *lda*, linear discriminant analysis; *qda*, quadratic discriminant analysis; *lr*, logistic regression; *ct*, conditional inference tree; *knn*, *k*-nearest neighbors. **Voting ensemble contains support vector machine, elastic net, and random forest.*

3. Results

<u>Table 2</u>: Mean Accuracy for 12 Models

Rank	Model	Kappa
1	voting ensemble	**0.9601819**
2	support vector machine	**0.9523703**
3	elastic net	**0.9521549**
4	extremely randomized trees	**0.9314840**
5	neural network	**0.9305945**
6	XGBoost	**0.9299473**
7	random forest	**0.9171861**
8	linear discriminant analysis	**0.9132007**
9	quadratic discriminant analysis	**0.9026230**
10	logistic regression	**0.8773293**
11	conditional inference tree	**0.8572459**
12	*k*-nearest neighbors	**0.8505412**

The two base learners with the lowest error were the support vector machine and elastic net (**Table 2**). The base learner with the greatest error was *k*-nearest neighbors. Some models, including the neural network, had inconsistent performance as illustrated by the spread of values (**Figure 1**). The random forest was the most consistent, with less dispersion than any other method.

The best simple voting ensemble was a combination of support vector machine, elastic net, and random forest. This was a surprise, since these were the #1, #2, and #6 base learners, respectively. It is not clear why the random forest was a good tiebreaker. It is reasonable to suspect the random forest made predictions that were not highly correlated with those of the other two models, so it contributed something invaluable to the ensemble.

Of the 1054 ensembles evaluated by the method, 191 had an average accuracy greater than 95% (**Table 3**).

All the models in this group seemed to include support vector machines, elastic net, or both.

Table 3: 15 Best Voting Ensembles (“1” indicates a base learner is present, “NA” indicates absence)

	ct	knn	rf	xt	xgb	qda	lda	nn	glm	en	svm	kappa
1	NA	NA	1	NA	NA	NA	NA	NA	NA	1	1	0.96186
2	NA	NA	NA	1	NA	NA	NA	NA	NA	1	1	0.96007
3	NA	NA	NA	NA	1	NA	NA	NA	NA	1	1	0.95992
4	NA	NA	1	NA	1	NA	NA	NA	1	1	1	0.95946
5	NA	1	NA	NA	NA	NA	1	1	NA	1	1	0.95941
6	NA	NA	NA	NA	1	NA	1	1	NA	1	1	0.95929
7	NA	NA	1	NA	1	NA	1	1	1	1	1	0.95871
8	NA	NA	1	NA	NA	NA	1	1	NA	1	1	0.95854
9	NA	NA	NA	1	1	NA	1	1	1	1	1	0.95843
10	NA	NA	1	1	NA	NA	NA	NA	1	1	1	0.95813
11	NA	NA	1	1	NA	NA	1	1	1	1	1	0.95812
12	1	NA	NA	NA	NA	NA	NA	NA	NA	1	1	0.95809
13	NA	NA	NA	NA	1	1	1	1	1	1	1	0.95802
14	NA	NA	NA	1	1	NA	NA	NA	1	1	1	0.95797
15	NA	1	NA	NA	NA	NA	NA	NA	NA	1	1	0.95768

What if I never included support vector machines and elastic net? Even after exclusing these models I still found 30 ensembles surpassing 94% accuracy. Once again, these ensembles considerably outperformed the best base learner in the system, extremely randomized trees (93.1% accuracy).

The best voting ensemble in this restricted group had 94.95% accuracy (**Table 4**) and combined 5 different base learners: extremely randomized trees, XGBoost, linear discriminant analysis, neural network, and logistic regression. As before, certain base learners in the system had mediocre performance individually.

Table 4: 15 Best Voting Ensembles Not Including Support Vector Machine or Elastic Net

	ct	knn	rf	xt	xgb	qda	lda	nn	glm	en	svm	kappa
1	NA	NA	NA	1	1	NA	1	1	1	NA	NA	0.94953
2	NA	1	NA	1	1	1	1	1	1	NA	NA	0.94667
3	NA	NA	1	NA	1	NA	1	1	1	NA	NA	0.94632
4	NA	1	1	1	1	NA	1	1	1	NA	NA	0.94499
5	NA	NA	1	1	1	1	1	1	1	NA	NA	0.94459
6	NA	1	NA	1	1	NA	NA	1	1	NA	NA	0.94455
7	NA	NA	NA	1	1	NA	NA	NA	1	NA	NA	0.94396
8	NA	1	NA	1	1	NA	1	1	NA	NA	NA	0.94346
9	NA	NA	NA	1	1	1	1	NA	1	NA	NA	0.94345
10	NA	NA	1	NA	1	NA	NA	NA	1	NA	NA	0.94325
11	NA	1	1	NA	1	1	1	1	1	NA	NA	0.94303
12	NA	NA	1	1	1	1	NA	NA	1	NA	NA	0.94271
13	1	NA	NA	1	1	1	1	1	1	NA	NA	0.94260
14	NA	1	1	1	1	1	NA	1	1	NA	NA	0.94256
15	NA	NA	1	1	1	NA	1	NA	1	NA	NA	0.94253

5. Discussion / Conclusion

In this paper I have proposed a supervised method of preparing simple voting ensembles. The method relies on tuning a set of learners and aggregating predictions in different combinations. The best model minimizes the expected loss $\mathbb{E}[1 - \kappa(X, \Psi(F))]$ averaged across several cycles of k-fold cross-validation. In at least two cases, simple voting ensembles convincingly showed the best performance (**Table 2** and **Table 3**). While the experiments in this paper involve binary classification, the method should, in theory, generalize to other loss functions and even regression.

Under ideal conditions, the method can test hundreds of ensembles automatically. In fact, the runtime for generating **Figure 1** is only about 149 seconds. The scalability of the method is bounded chiefly by the size of the data set, the number of cross-validation cycles, and the number of models considered. (Here, the most expensive model was the support vector machine, which utilized a polynomial kernel.) Since the number of voting ensembles evaluated is about $(2^L - 1)/2$, the runtime increases exponentially when we involve a large set of base learners. Tuning models individually can also be expensive. However, parallel programming and supervised parameter searches can come to the rescue, and there are algorithms like fast cross-validation via sequential testing (2015) that use clever tricks to speed up cross-validation.

Having performance estimates for multiple ensembles means the ensembles themselves can be combined. For example, it is possible to train multiple ensembles each specializing on a different transformation of the data: ensemble A trains on the original data, ensemble B trains on the principal components, ensemble C trains on derived features, etc. It is tempting to wonder if such “meta-ensembles” might be successful.

My method echoes the “Super Learner” framework first developed by van der Laan et al. (2007) and revisited in subsequent papers (Polley et al. 2010). However, my method is not quite the same. While the Super Learner prepares just one ensemble from a library of base learners, my method prepares many ensembles ranked by their generalization error. And Super Learner uses a weighted voting scheme, while my method does not. I suspect that weighted voted ensembles might be better than simple voting ensembles, and I would like to test this in the future.

Another good research question is understanding why the best ensemble (**Table 3**, Line 1) recruited a random forest instead of another model. In theory, the answer has to do with diversity, but the mechanism is unclear.

6. Appendix

Supplementary Table 1: Complete description of features. Lifted from Street, Wolberg, and Mangasarian, 1993.

Feature	Description
1. Radius	The radius of an individual nucleus is measured by averaging the length of the radial line segments defined by the centroid of the snake and the individual snake points.
2. Perimeter	The total distance between the snake points constitutes the nuclear perimeter.
3. Area	Nuclear area is measured simply by counting the number of pixels on the interior of the snake and adding one-half of the pixels in the perimeter.
4. Compactness	Perimeter and area are combined to give a measure of the compactness of the cell nuclei using the formula perimeter^2/area . This dimensionless number is minimized by a circular disk and increases with the irregularity of the boundary. However, this measure of shape also increases for elongated cell nuclei, which do not necessarily indicate an increased likelihood of malignancy. The feature is also biased upward for small cells because of the decreased accuracy imposed by digitization of the sample. We compensate for the fact that no single shape measurement seems to capture the idea of “irregular” by employing several different shape features.
5. Smoothness	The smoothness of a nuclear contour is quantified by measuring the difference between the length of a radial line and the mean length of the lines surrounding it. This is similar to the curvature energy computation in the snakes.
6. Concavity	In a further attempt to capture shape information we measure the number and severity of concavities or indentations in a cell nucleus. We draw chords between non-adjacent snake points and measure the extent to which the actual boundary of the nucleus lies on the inside of each chord. This parameter is greatly affected by the length of these chords, as smaller chords better capture small concavities. We have chosen to emphasize small indentations, as larger shape irregularities are captured by other features.
7. Concave Points	This feature is similar to Concavity but measures only the number, rather than the magnitude, of contour concavities.
8. Symmetry	In order to measure symmetry, the major axis, or longest chord through the center, is found. We then measure the length difference between lines perpendicular to the major axis to the cell boundary in both directions. Special care is taken to account for cases where the major axis cuts the cell boundary because of a concavity.
9. Fractal Dimensions	The fractal dimension of a cell is approximated using the “coastline approximation” described by Mandelbrot. The perimeter of the nucleus is measured using increasingly larger 'rulers'. As the ruler size increases, decreasing the precision of the measurement, the observed perimeter decreases. Plotting these to values on a log scale and measuring the downward slope gives (the negative of) an approximation to the fractal dimension. As with all the shape features, a higher value corresponds to a less regular contour and thus to a higher probability of malignancy.
10. Texture	The texture of the cell nucleus is measured by finding the variance of the gray scale intensities in the component pixels.

In addition to these 10 features, 20 other features were derived that were functions of these 10; specifically the mean and the most extreme value for each feature from the set of sample images.

Supplementary Table 2: Model Parameters.

Model	Library	Function	Parameters
1. support vector machine	`e1071`	`svm`	`kernel = "polynomial"` `degree = 3` `coef0 = 1` `epsilon = 0.001` `scale = TRUE`
2. elastic net	`glmnet`	`glmnet`	`alpha = 0.15` `lambda = 0.003110412` `family = "binomial"`
3. neural network	`neuralnet`	`neuralnet`	`hidden = 35` `algorithm = rprop+` `stepmax = 1e7` `linear.output = FALSE` `threshold = 0.000001`
4. extremely randomized trees	`ranger`	`ranger`	`mtry = 15` `max.depth = 27` `num.trees = 1000` `splitrule = "extratrees"` `replace = F` `sample.fraction = 1`
5. XGBoost	`xgboost`	`xgboost`	`booster = "gbtree"` `eta = 0.3` `gamma = 0` `subsample = 0.8` `colsample_by_tree = 0.4` `max_depth = 14` `nrounds = 223`
6. random forest	`ranger`	`ranger`	`mtry = 10` `max.depth = 37` `num.trees = 1000` `replace = TRUE` `sample.fraction = 0.632`
7. linear discriminant analysis	`MASS`	`lda`	`prior = c(0.5, 0.5)`
8. quadratic discriminant analysis	`MASS`	`qda`	`prior = c(0.5, 0.5)`
9. logistic regression	`stats`	`glm`	`family = "binomial"` `maxit = 1000`
10. conditional inference tree	`partykit`	`ctree`	default parameters
11. *k*-nearest neighbors	`class`	`knn`	`k = 11`

Inputs were centered and scaled for two algorithms: support vector machines (by setting `scale = TRUE`) and neural networks (by passing the scaled data matrix to the function call). Otherwise, to ensure fair comparisons, there was no preprocessing of features.

Works Cited

Aleryani, Aliya, et al. “Multiple Imputation Ensembles (MIE) for Dealing with Missing Data.” *SN Computer Science*, vol. 1, no. 3, 2020, doi:10.1007/s42979-020-00131-0.

Bian, Yijun, and Huanhuan Chen. “When Does Diversity Help Generalization In Classification Ensembles?” *IEEE Transactions on Cybernetics*, 30 Oct. 2019.

Breiman, Leo. “Bagging Predictors.” *Machine Learning*, vol. 24, no. 2, Aug. 1996, pp. 123–140., doi:10.1007/bf00058655.

Breiman, Leo. “Random Forests.” *Machine Learning*, vol. 45, no. 1, Oct. 2001, pp. 5–32., doi:10.1023/a:1010933404324.

Cohen, Jacob. “A Coefficient of Agreement for Nominal Scales.” *Educational and Psychological Measurement*, vol. 20, no. 1, 1 Apr. 1960, pp. 37–46., doi:10.1177/001316446002000104.

Friedman, Jerome H. “Greedy Function Approximation: A Gradient Boosting Machine.” *The Annals of Statistics*, vol. 29, no. 5, Feb. 1999, pp. 1189–1232., doi:10.1214/aos/1013203451.

Friedman, Jerome H. “Stochastic Gradient Boosting.” *Computational Statistics & Data Analysis*, vol. 38, no. 4, Feb. 2002, pp. 367–378., doi:10.1016/s0167-9473(01)00065-2.

Geurts, Pierre, et al. “Extremely Randomized Trees.” *Machine Learning*, vol. 63, no. 1, 2 Mar. 2006, pp. 3–42., doi:10.1007/s10994-006-6226-1.

Gu, Shenkai, et al. “Multi-Objective Ensemble Generation.” *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 5, no. 5, 7 July 2015, pp. 234–245., doi:10.1002/widm.1158.

Hastie, Trevor, et al. *The Elements of Statistical Learning Data Mining, Inference, and Prediction*. Springer New York, 2001.

Herrera, Francisco, et al. “Ensemble-Based Classifiers.” *Multilabel Classification Problem Analysis, Metrics and Techniques*, by Francisco Herrera et al., Springer International Publishing, 2016, pp. 101–113.

Kohavi, Ron. “A study of cross-validation and bootstrap for accuracy estimation and model selection”. *Proceedings of the International Joint Conference on Artificial Intelligence*, August 1995, pp. 1137–1145.

Krueger, Tammo, et al. “Fast Cross-Validation via Sequential Testing.” *Journal of Machine Learning Research*, vol. 16, June 2015, pp. 1103–1155.

Kuncheva, Ludmila I., and Christopher J. Whitaker. “Measures of Diversity in Classifier Ensembles and Their Relationship with the Ensemble Accuracy.” *Machine Learning*, vol. 51, no. 2, May 2003, pp. 181–207., doi:10.1023/a:1022859003006.

Liu, Y., and X. Yao. “Ensemble Learning via Negative Correlation.” *Neural Networks*, vol. 12, no. 10, 5 July 1999, pp. 1399–1404., doi:10.1016/s0893-6080(99)00073-8.

Polley, Eric C., and van der Laan, Mark J. “Super Learner in Prediction.” *U.C. Berkeley Division of Biostatistics Working Paper Series*, May 2010. Working Paper 266.

Rokach, Lior. “Ensemble-Based Classifiers.” *Artificial Intelligence Review*, vol. 33, no. 1-2, 19 Nov. 2009, pp. 1–39., doi:10.1007/s10462-009-9124-7.

Srivastava, Nitish et al. “Dropout: A Simple Way to Prevent Neural Networks from Overfitting.” *Journal of Machine Learning Research*, vol. 15, June 2014, pp. 1929–1958. Print.

Street, W. N., et al. “Nuclear Feature Extraction for Breast Tumor Diagnosis.” *Biomedical Image Processing and Biomedical Visualization*, 29 July 1993, doi:10.1117/12.148698.

Tukey, John Wilder. *Exploratory Data Analysis*. Addison-Wesley, 1970.

van der Laan, Mark J., et al. “Super Learner.” *Statistical Applications in Genetics and Molecular Biology*, vol. 6, no. 1, 16 Jan. 2007, doi:10.2202/1544-6115.1309.

Vanwinckelen, Gitte, and Blockeel, Henri. “On Estimating Model Accuracy with Repeated Cross-Validation.” *BeneLearn 2012: Proceedings of the 21st Belgian-Dutch Conference on Machine Learning*, 2012, pp. 39–44.

Wolpert, David H. “Stacked Generalization.” *Neural Networks*, vol. 5, no. 2, Jan. 1992, pp. 241–259., doi:10.1016/s0893-6080(05)80023-1.

Wu, Xindong, et al. “Top 10 Algorithms in Data Mining.” *Knowledge and Information Systems*, vol. 14, no. 1, 4 Dec. 2007, pp. 1–37., doi:10.1007/s10115-007-0114-2.

Yu, Yang et al. “Diversity Regularized Machine.” *IJCAI*, p. 1603, 2011.

Zhou, Zhi-Hua, et al. “Ensembling Neural Networks: Many Could Be Better than All.” *Artificial Intelligence*, vol. 137, no. 1-2, May 2002, pp. 239–263., doi:10.1016/s0004-3702(02)00190-x.