MODEL PERFORMANCE EVALUATION AND IMPROVEMENT

Objectives:

- Evaluating various ML algorithms as per the performance metric.
- Improving performance by parameter tuning, performance measurements
- Ameliorating the overall performance by Ensembles: Bagging, Boosting and and Random forests.

Model Performance

- Model performance should be evaluated not just by predictive accuracy but other performance measures obtained by the different methods.
- In regards to classification we evaluate the classification accuracy by the measure
  Number of correct predictions/total predictions.
  If accuracy obtained is 99.99% then we have an error rate of .01%
  The ML will give us 10 errors in 10,000 records therefore if we are evaluating a scenario which is pertaining cancer or genetic disorders then we should not totally depend on this prediction.
- The best measure to utilize is a measure that ascertains whether the algorithm has achieved its purpose.
- We would like to determine if the classifier will extrapolate well for future cases and since we do not really have the futuristic data we simulate the potential cases. By implementing this process we can evaluate the strength and weakness of the specific learner.
- From prior knowledge we know we can obtain the actual class values as well as predicted class values. The classification and prediction is measured using these two measures.
- An additional measure that we can obtain is how probable a particular classification is for a specific observation/example.
- Each predicted class value is associated with a probability for this classification.
- The predicted probabilities will help determine the confidence of the prediction
- Most of the classification algorithms use the method predict() for comparing actual and predicted values. To obtain the predicted class value the parameter type = class has to be set and additionally to obtain the probability for this assignment the type parameter is set to “prob”, ”posterior” or “raw” etc depending on the classifier.

EXAMPLE

```
> sms_test_prob <- predict(sms_classifier, sms_test, type = "raw")
> head(sms_test_prob)
                ham         spam
[1,] 9.999995e-01 4.565938e-07
[2,] 9.999995e-01 4.540489e-07
[3,] 9.998418e-01 1.582360e-04
[4,] 9.999578e-01 4.223125e-05
[5,] 4.816137e-10 1.000000e+00
[6,] 9.997970e-01 2.030033e-04
```

```
# combine the results into a data frame
sms_results <- data.frame(actual_type = sms_test_labels,
                          predict_type = sms_test_pred,
                          prob_spam = round(sms_test_prob[ , 2], 5),
                          prob_ham = round(sms_test_prob[ , 1], 5))
```

The actual values(from the target feature), predicted values prob of ham and prob of spam can be collected and stored in a data frame called sms_results

These are accurate predictions and the probabilities also align with the prediction.

```
> head(sms_results)
  actual_type predict_type prob_spam prob_ham
1         ham          ham   0.00000  1.00000
2         ham          ham   0.00000  1.00000
3         ham          ham   0.00016  0.99984
4         ham          ham   0.00004  0.99996
5        spam         spam   1.00000  0.00000
6         ham          ham   0.00020  0.99980
```

Unlike the former predictions the following prediction are inaccurate and these probabilities are close to 50%:

```
> head(subset(sms_results, prob_spam > 0.40 & prob_spam < 0.60))
     actual_type predict_type prob_spam prob_ham
377         spam          ham   0.47536  0.52464
717          ham         spam   0.56188  0.43812
1311         ham         spam   0.57917  0.42083
```

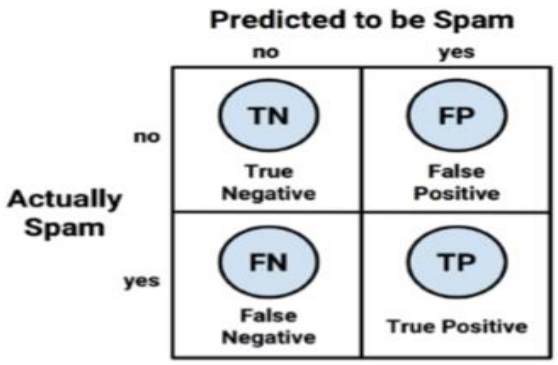
The following are inaccurate even though the probabilities of these predictions are high:

```
> head(subset(sms_results, actual_type != predict_type))
    actual_type predict_type prob_spam prob_ham
```

```
53          spam          ham   0.00071  0.99929
59          spam          ham   0.00156  0.99844
73          spam          ham   0.01708  0.98292
76          spam          ham   0.00851  0.99149
184         spam          ham   0.01243  0.98757
332         spam          ham   0.00003  0.99997
```

The predictions can be accurate or inaccurate therefore to evaluate if the overall model is a value addition we use measures like Confusion Matrix.

- **True Positive (TP)**: Correctly classified as the class of interest
- **True Negative (TN)**: Correctly classified as not the class of interest
- **False Positive (FP)**: Incorrectly classified as the class of interest
- **False Negative (FN)**: Incorrectly classified as not the class of interest



Accuracy of the prediction can be defined by the measure accuracy or success rate. This measure the proportion of correctly classified examples.

$$\text{accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}$$

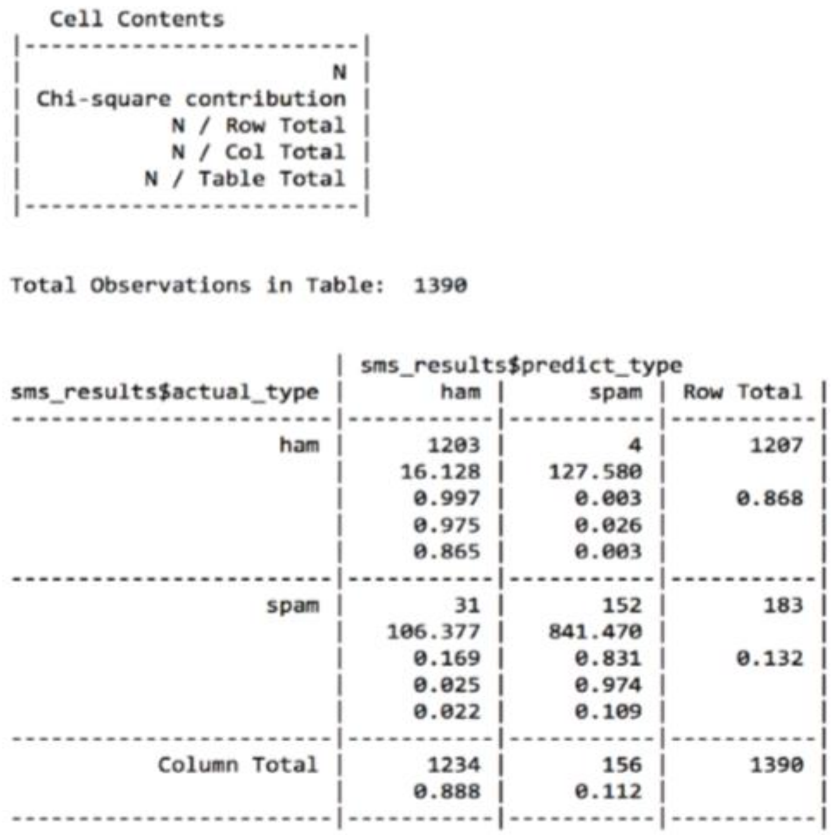
And the proportion of incorrectly classified examples are measured by the error rate:

$$\text{error rate} = \frac{\text{FP} + \text{FN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} = 1 - \text{accuracy}$$

To create the Confusion matrix the following command has to be used:

```
> library(gmodels)
> CrossTable(sms_results$actual_type, sms_results$predict_type)
```

The result is a confusion matrix with a wealth of additional detail:



```
   Cell Contents
|-------------------------|
|                       N |
| Chi-square contribution |
|           N / Row Total |
|           N / Col Total |
|         N / Table Total |
|-------------------------|


Total Observations in Table:  1390


                        | sms_results$predict_type
sms_results$actual_type |       ham |      spam | Row Total |
------------------------|-----------|-----------|-----------|
                    ham |      1203 |         4 |      1207 |
                        |    16.128 |   127.580 |           |
                        |     0.997 |     0.003 |     0.868 |
                        |     0.975 |     0.026 |           |
                        |     0.865 |     0.003 |           |
------------------------|-----------|-----------|-----------|
                   spam |        31 |       152 |       183 |
                        |   106.377 |   841.470 |           |
                        |     0.169 |     0.831 |     0.132 |
                        |     0.025 |     0.974 |           |
                        |     0.022 |     0.109 |           |
------------------------|-----------|-----------|-----------|
           Column Total |      1234 |       156 |      1390 |
                        |     0.888 |     0.112 |           |
------------------------|-----------|-----------|-----------|
```

- The accuracy and error rates can be ascertained from the confusion matrix as well.
- There are countless measures to evaluate performance .The caret package by Max Kuhn provides several measures to evaluate and improve performance.
- The caret package provides a function that detects these performance metrics:

```
> library(caret)
> confusionMatrix(sms_results$predict_type,
    sms_results$actual_type, positive = "spam")
```

This results in the following output:

```
Confusion Matrix and Statistics

          Reference
Prediction  ham spam
      ham  1203   31
      spam    4  152

               Accuracy : 0.9748
                 95% CI : (0.9652, 0.9824)
    No Information Rate : 0.8683
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.8825
 Mcnemar's Test P-Value : 1.109e-05

            Sensitivity : 0.8306
            Specificity : 0.9967
         Pos Pred Value : 0.9744
         Neg Pred Value : 0.9749
             Prevalence : 0.1317
         Detection Rate : 0.1094
   Detection Prevalence : 0.1122
      Balanced Accuracy : 0.9136

       'Positive' Class : spam
```

Kappa statistics adjusts the accuracy of the prediction taking into account the possibility for the correct prediction to be obtained by chance only. Kappa statistics can range from 0 to 1.

- Poor agreement = less than 0.20
- Fair agreement = 0.20 to 0.40
- Moderate agreement = 0.40 to 0.60
- Good agreement = 0.60 to 0.80
- Very good agreement = 0.80 to 1.00

Value of one means perfect agreement of predicted and true value.

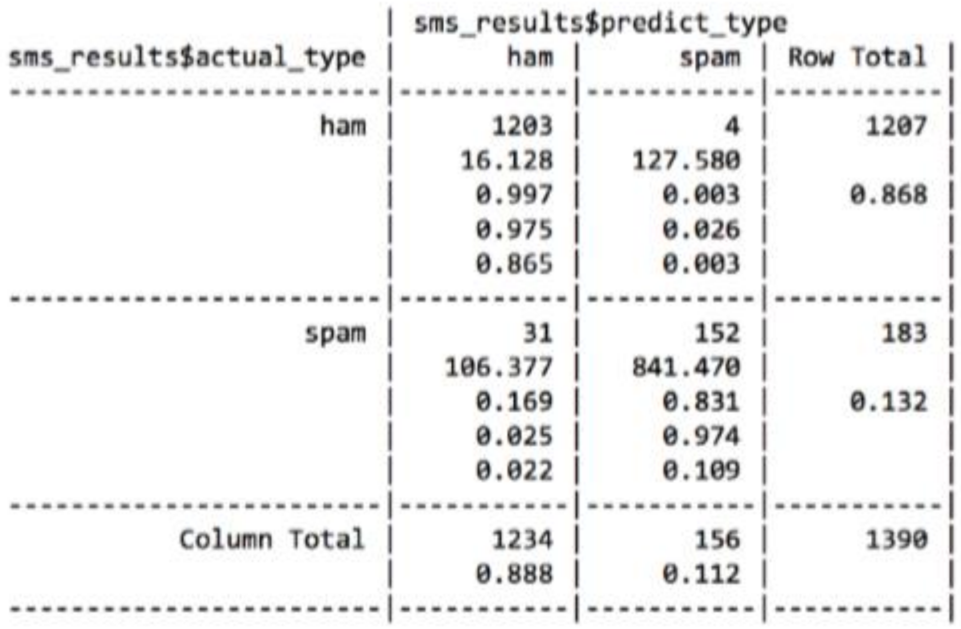
Value less than one upto zero is imperfect agreement of predicted and true value.

Kappa Statistics Formula:

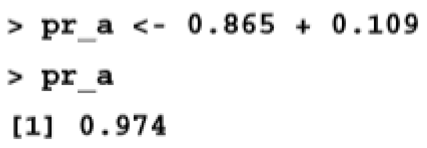
$$\kappa = \frac{\Pr(a) - \Pr(e)}{1 - \Pr(e)}$$

Pr(a) is the proportion of actual agreement

Pr(e) is the expected agreement between classifier and true value taking the assumption that they are chosen at random.



```
                       | sms_results$predict_type
sms_results$actual_type |       ham |      spam | Row Total |
-----------------------|-----------|-----------|-----------|
                   ham |      1203 |         4 |      1207 |
                       |    16.128 |   127.580 |           |
                       |     0.997 |     0.003 |     0.868 |
                       |     0.975 |     0.026 |           |
                       |     0.865 |     0.003 |           |
-----------------------|-----------|-----------|-----------|
                  spam |        31 |       152 |       183 |
                       |   106.377 |   841.470 |           |
                       |     0.169 |     0.831 |     0.132 |
                       |     0.025 |     0.974 |           |
                       |     0.022 |     0.109 |           |
-----------------------|-----------|-----------|-----------|
          Column Total |      1234 |       156 |      1390 |
                       |     0.888 |     0.112 |           |
-----------------------|-----------|-----------|-----------|
```



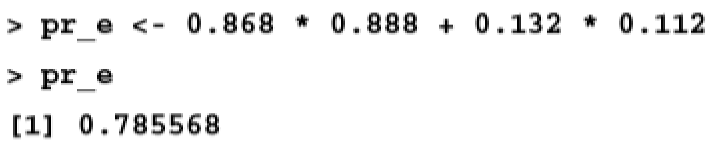
```
> pr_a <- 0.865 + 0.109
> pr_a
[1] 0.974
```

The observed and actual values agree 97.4% times of the times. This is equal to the accuracy.

Now evaluating the expected agreement Pr(e) which is the probability that the predicted and actual values match by chance.

The probability of these events can be found by the calculations

P(Actual type is ham)*P(Predicted type is ham)+ P(Actual type is spam)*P(Predicted type is spam)



```
> pr_e <- 0.868 * 0.888 + 0.132 * 0.112
> pr_e
[1] 0.785568
```

That means that the actual and observed agree just by chance alone 78.6% of the times.

```
> k <- (pr_a - pr_e) / (1 - pr_e)
> k
[1] 0.8787494
```

Using the Kappa interpretation we can see that the actual and the predicted values are in very good agreement.

SENSTIVITY (True Positive rate) Measures the proportion of positive example that were correctly classified

$$\text{sensitivity} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

SPECIFICITY (True Negative Rate) Measures the proportion of negative examples that were correctly classified

$$\text{specificity} = \frac{\text{TN}}{\text{TN} + \text{FP}}$$

```
> sens <- 152 / (152 + 31)
> sens
[1] 0.8306011
```

Similarly, for specificity we can calculate:

```
> spec <- 1203 / (1203 + 4)
> spec
[1] 0.996686
```

The closer specificity and sensitivity is to 1 the better the prediction.

Senstivity of 83.1% implies that 83.1% of the spam message were correctly classified.

The specificity is very high =99.7% that implies that we have misclassified .3% ie we have classified .3% of the ham(valid) emails to spam. This prediction might not be acceptable to us or the tradeoff might be acceptable given that the majority of the ham emails were correctly classified.

Different models are created and then compared to reach the optimal threshold for specificity and sensitivity.

Precision : PPV Positive Predictive Value

Proportion of positive examples that are truly positive. This informs us how often we reject spam and not ham.

$$\text{precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

Recall : Measures how complete the results are. This is the same as sensitivity. If the recall is higher, the learner captures higher number of positive cases ie majority of spam messages were classified accurately.

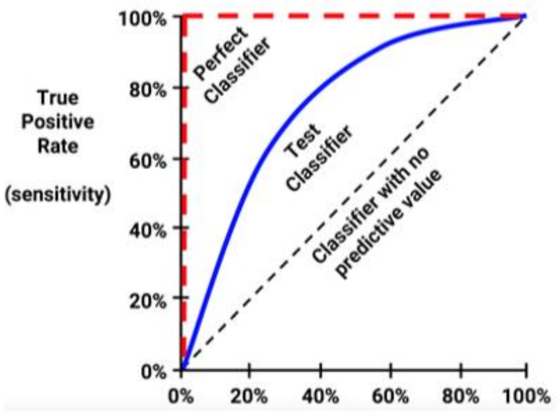
$$\text{recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

F Measure Combines the recall and precision in one measure by an average called Harmonic mean.

$$\text{F-measure} = \frac{2 \times \text{precision} \times \text{recall}}{\text{recall} + \text{precision}} = \frac{2 \times \text{TP}}{2 \times \text{TP} + \text{FP} + \text{FN}}$$

This measure uses the harmonic mean (rate of changes) and this can be used to compare numerous models as long as the weightages to precision and recall are the same..

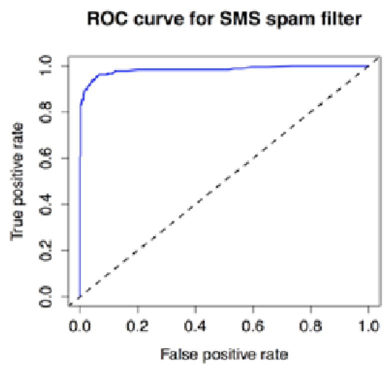
Visualization of Performances ROC Receiver Operating Characteristics curve helps examine tradeoff between detection of true positives (sensitivity on y axis) while avoiding false positives (1-specificity on x axis).



The red vertical line depicting the 100% accuracy. Area under ROC (AUC) ranges from .5 to 1 (Perfect classifier). Area of .5 has no classification accuracy.

Convention for ROC AUC:

- A: Outstanding = 0.9 to 1.0
- B: Excellent/good = 0.8 to 0.9
- C: Acceptable/fair = 0.7 to 0.8
- D: Poor = 0.6 to 0.7
- E: No discrimination = 0.5 to 0.6

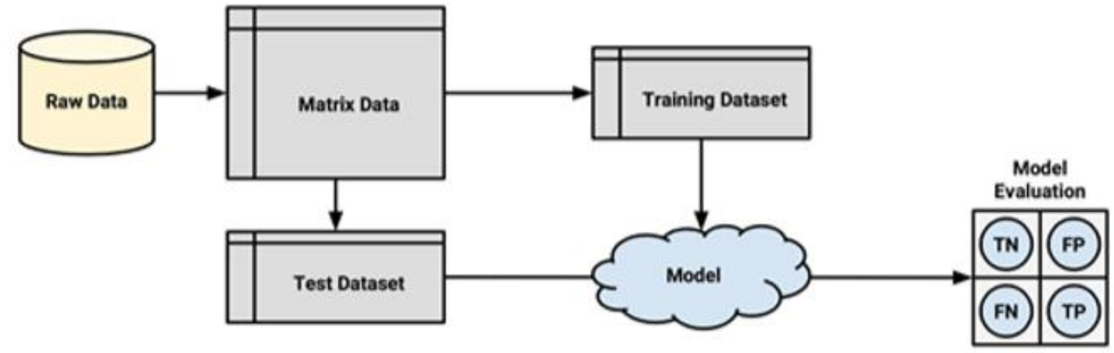


AUC =98%

Estimation of Future Performance

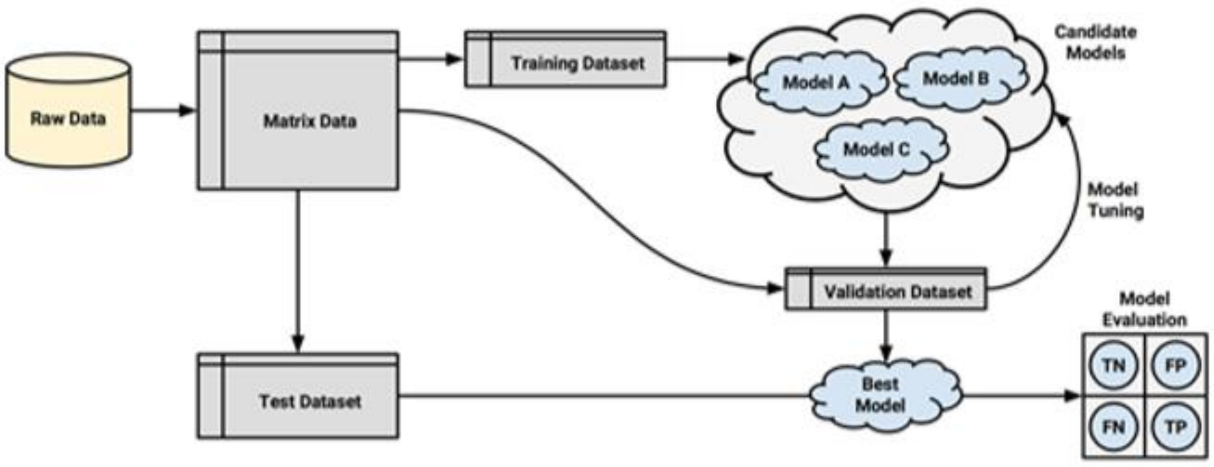
The re-substitution error method of evaluating how well the training data gets classified is not a good diagnostic method since it is usually not generalizable to the future data.

The hold out method of dividing the data into training and testing dataset and then predicting the how well the test data generalizes. Typically 1/4 of the data is used for testing and 3/4 of the data is used for training but can vary depending on the final data size.



If several models are run on the test data and the model giving the best accuracy is chosen then selecting the one with the highest accuracy is a biased measure of performance on unseen data.

To handle this issue the original is divided into validation set, training set and testing set. The validation data set is used to for iterating and improving the model and then finally the test data is used for final prediction.



Typically 50% is used for testing and 25% is used for testing and 25% for validation.

Cross Validation

Repeated hold out method is K fold cross validation divides data into k random partitions. Using k=10 is used since empirically it has been proved that this is big enough. 9 of the folds are used for training and the 10th fold is used for testing. This is redone 10 times with another partition being the validation set. After this the average performance of model is reported.

Bootstrap Sampling

Another method that is used as an alternate method instead of cross validation. Boot strapping sampling implies the creation of random subsets of training and testing sets and then the performance of these are averaged across all cases. The sampling is done with replacement therefore a sample can have the same elements. For the k fold validation the partitions have distinct elements but for the bootstrap sampling the elements can be repeated.

Improving Performance:

The Strategies for improving model performances are as follows:

1) The performance of the model can be maximized by systematically implementing search of optimal training conditions by parametric tuning.
2) Implementing a cluster of learners in conjunction with each other to improve the overall performance to offset the tradeoffs in context to the individual algorithm's strength and weakness.
3) To usage of repertoire of decisions trees which showcases exemplary performance.

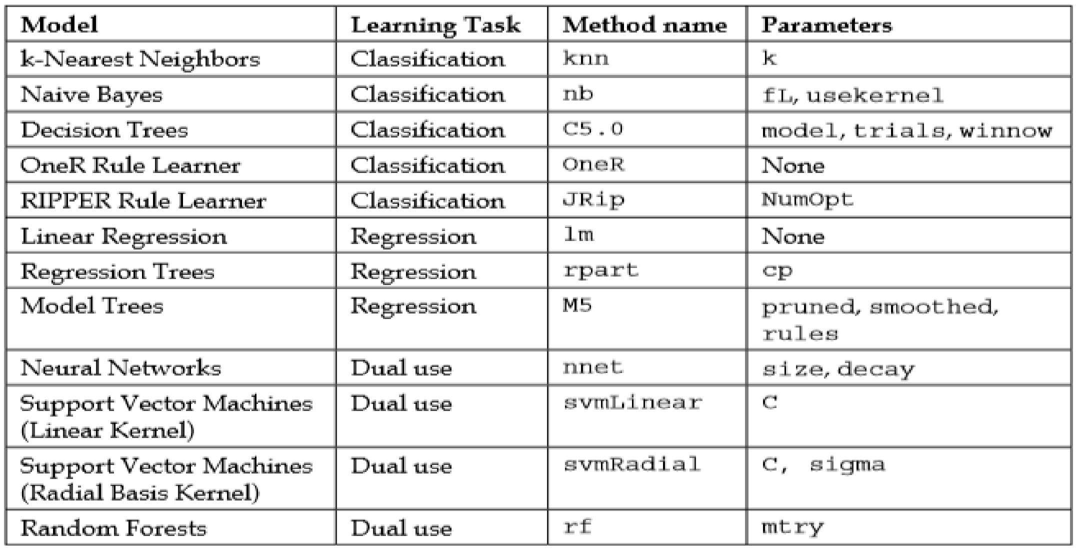
To improve model performance at least one of the above strategies has to be implemented. These strategy have often been used for Kaggle competitions.

Improving Performance by Parameter Tuning

To start the parameter tuning process following aspects need to be determined:

1) Which machine learning model can be be trained on the data given the nature of the problem?
2) Contextual to the specific machine learning algorithm which tuning parameters can be manipulated to obtain the best performance.
3) What is the criteria that is being targeted to find the best model for problem?

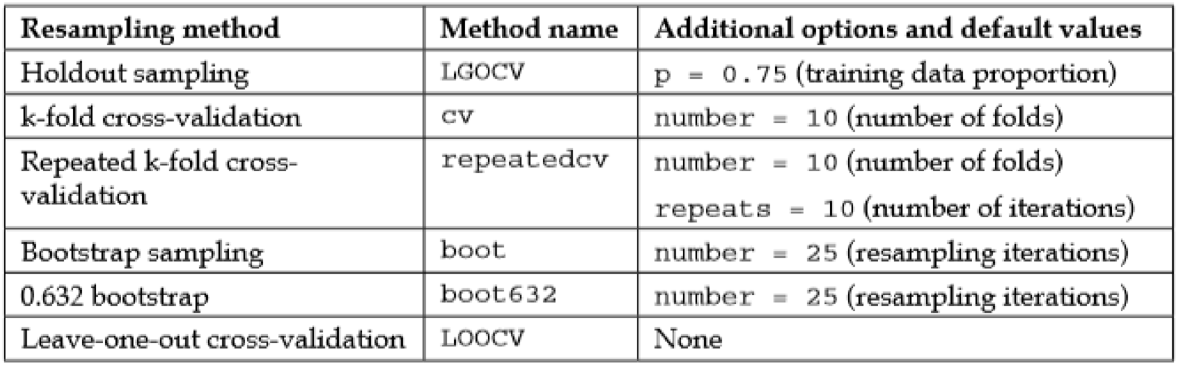
An algorithm can be improved by changing some parameter like number of nearest neighbors to run KNN etc. Some of these features have been listed in the table below. A modelLookup() function in caret showcases some of the parameters that can be tuned for an algorithm.



Model	Learning Task	Method name	Parameters
k-Nearest Neighbors	Classification	`knn`	`k`
Naive Bayes	Classification	`nb`	`fL`, `usekernel`
Decision Trees	Classification	`C5.0`	`model`, `trials`, `winnow`
OneR Rule Learner	Classification	`OneR`	None
RIPPER Rule Learner	Classification	`JRip`	`NumOpt`
Linear Regression	Regression	`lm`	None
Regression Trees	Regression	`rpart`	`cp`
Model Trees	Regression	`M5`	`pruned`, `smoothed`, `rules`
Neural Networks	Dual use	`nnet`	`size`, `decay`
Support Vector Machines (Linear Kernel)	Dual use	`svmLinear`	`C`
Support Vector Machines (Radial Basis Kernel)	Dual use	`svmRadial`	`C, sigma`
Random Forests	Dual use	`rf`	`mtry`

The third step for parameter tuning is involves identifying the methodology of Evaluating Model performance by selecting the resampling technique(bootstrap, cross validation) and also deciding upon the measure to use for ascertaining the prediction accuracy. Caret selects the model showcasing the best accuracy as per the choice of performance metric By default caret provides the prediction accuracy based of the bootstrap sampling .

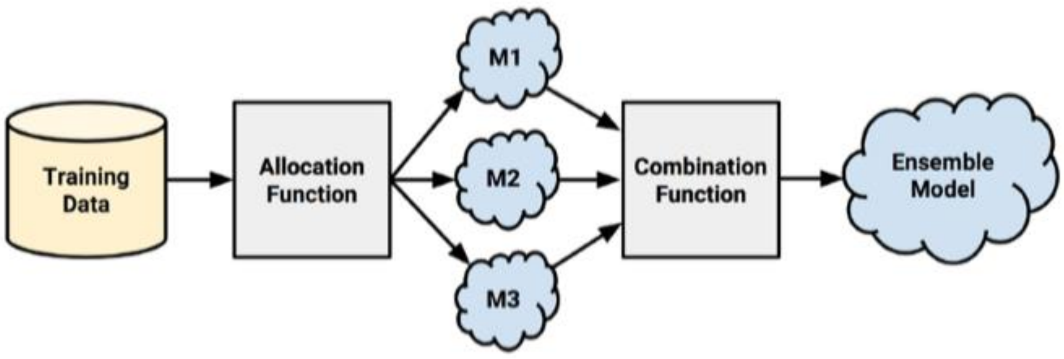
trainControl() function in the caret package is used to create a set of configuration and this is available as a control object that guides the train function to use the resampling methods



Resampling method	Method name	Additional options and default values
Holdout sampling	`LGOCV`	`p = 0.75` (training data proportion)
k-fold cross-validation	`cv`	`number = 10` (number of folds)
Repeated k-fold cross-validation	`repeatedcv`	`number = 10` (number of folds) `repeats = 10` (number of iterations)
Bootstrap sampling	`boot`	`number = 25` (resampling iterations)
0.632 bootstrap	`boot632`	`number = 25` (resampling iterations)
Leave-one-out cross-validation	`LOOCV`	None

Performance Improvement by Meta Learning

If the performance of the algorithms is increased by combining various algorithm then this methodology is called Meta Learning or Ensembles, By using a number of weak learners in conjunction with each other causes the creation of a strong learner. Allocation functions assigns the data to each algorithm. Bagging , Boosting and Random Forest are important algorithms.



Allocation Function Determines how much training data is allocated to each algorithm as well as the features to be used by it. For the model creation. For instance ensemble might use bootstrap sampling to create more differentiated sample as and assign them to the training algorithms. Ideally the algortihms incorporated in the ensemble should be diverse in nature for a wholistic classification or prediction.

Combination Function Implements the decision making functionality of how to reconcile the prediction conflicts amongst ML algorithms. For deciding upon the prediction/classification task a majority vote startegy as well as weighing each model based on prior performance could be implemented.

Ensemble helps in generalizability for futuristic scenarios. The bias of any specific algorithm does monopolize the final prediction.The overfitting is also reduced.

Performance improves on small data sets because multiple algortihms are run synergistically with resampling techniques like boot strapping. Since some models run out of memory due the complexity several small models can be run instead of one big model.

Real life problems are complicated therefore the fine grained and nuanced intricacies can be extracted y models that are segmented rather than a running a big model.

The combination function governs how disagreements among the predictions are reconciled. For example, the ensemble might use a majority vote to determine the final prediction, or it could use a more complex strategy such as weighting each model's votes based on its prior performance

Bagging

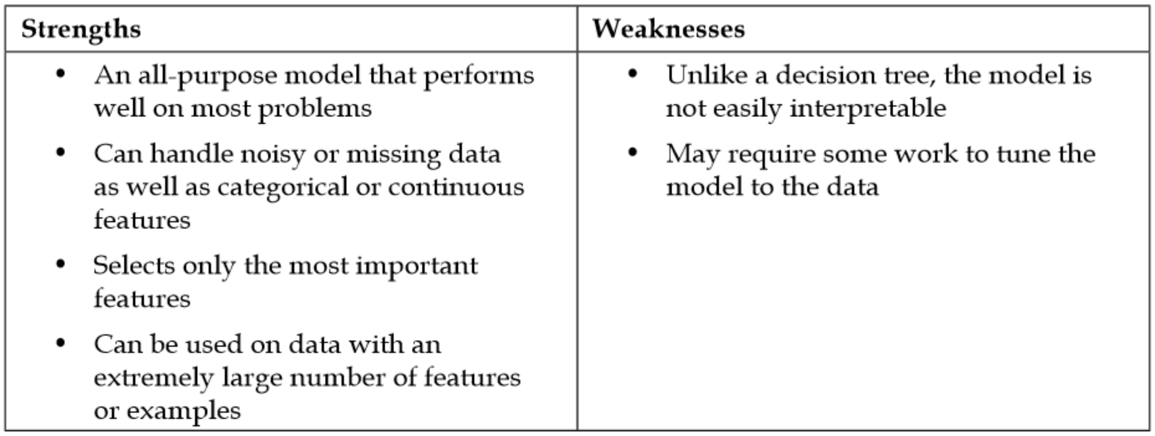
Bagging or Bootstrap aggregation implements boot strap sampling to generate training samples. The samples are individually run through a particular ML algorithms and finally the model classification is decided by the majority vote or averaging for the objective of prediction. This technique is very beneficial for scenarios where the algorithm is unstable in context to small changes in the input data. Trees have a propensity of being very sensitive to the input data therefore ensemble can be effectively used in this context. Ipred package provides the bagging implementation.

Boosting

Boosting like bagging generates resampled samples but it is used to create complementary learners (can be more than two)and higher voting weightage is attributed to the learners that a better prior performance. This mechanism ensures that weak learners perform as well as the strong learner if not better. AdaBoost is a tree based Boosting technique. This is considered to be a very innovative and novel discoveries in contemporary times.

Random Forest Ensemble:

This is an ensemble which is exclusively implementing decision trees. This technique imlements the paradigm of bagging in conjunction with random feature selection to build variegation in the model. Once the ensemble of trees is created the classification or predictions are made as per the votes or averages. Due it high performance Random forest ensemble is emerging as one of the most popular ML algorithm. The strengths and weakness for the Random forest can be enumerated as follows:



Strengths	**Weaknesses**
• An all-purpose model that performs well on most problems • Can handle noisy or missing data as well as categorical or continuous features • Selects only the most important features • Can be used on data with an extremely large number of features or examples	• Unlike a decision tree, the model is not easily interpretable • May require some work to tune the model to the data