# Random Forest Email Notes

## Q 29/11

[rf\_oob\_pred, rf\_oob\_scores] = oobPredict(RandomForest);

rf\_oob\_results = confusionmat(**y\_train**, double(cell2mat(rf\_oob\_pred)));

## Q&A 27/11

### Q\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

"Mdl = TreeBagger(NumTrees,Tbl,ResponseVarName) returns an ensemble of NumTrees bagged classification trees trained using the sample data in the table Tbl. ResponseVarName is the name of the response variable in Tbl."

And it follows by saying "To label out-of-bag observations, pass Mdl to oobPredict."

### A\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Training set is used during hyperparameter tuning to find the best model.

model = TreeBagger(. . . , Training, . . .).

With oobPredict(), you use the above trained model to predict classes on the out of bag observations stored in the model.

The testing set is used to see the error of the final model on unseen data. After training the final model using the found parameters, you can test this model using predict() and the testing set.

## Q 27/11

I run the following to train a model ( I change the number of trees and other parameters to see which gives lowest error)

rf\_Mdl4 = TreeBagger(n,train\_predictors,train\_response,'OOBPrediction','On',...

    'Method','classification','CategoricalPredictors',cat\_cols);

### A 27/11

As for your bagged model, here are some optional suggestions. Yes, first by setting e.g. predTable = table, you can use the trained classifier to predict the class labels of the test set pred = predict(mdl, X\_test) and append the column of predictions to the results table predTable = [predTable pred].   
  
Then when producing the ensemble (final) predictions, you can initialise an empty cell array for the ensemble predictions, ensemblePred = {}, n = size(predTable,1).   
  
Then, use can consider using a loop, e.g.:  
for i = 1:n  
   predSet = table2cell(predTable(i,:))  
   predStat = tabulate(predSet)  
   statTable = cell2table(predStat) -- convert the prediction  
   [val, idx] = max(statTable.predStat2)  
   pred = char(statTable.predStat1(idx)) -- convert to character array  
   ensemblePred{i} = pred – and save the ensemble prediction  
end  
  
and reshape the ensemble predictions cell array to be a column vector rather than a row vector (i.e. so it matches Y\_train) n = size(ensemblePred, 2) and finalPred = reshape(ensemblePred, n, 1). Then, you can analyse the predictive performance, confusionmat(Y\_test, finalPred, ...) etc.  
  
Btw, there is also a way to save best model in compact mode, just FYI:  
  
opt\_mdl1 = TreeBagger(...)  
oobError1 = opt\_mdl1.oobError('Mode','Ensemble')  
save best model in compact mode:  
mdlC = opt\_mdl1.compact  
and use saved model for prediction on test set:  
[predClass,classifScore] = mdlC.predict(X\_test)  
  
Best,  
Alex

## A 26/11

Hi Behzad,

For RF, no need to bootstrap manually.

Best regards,

Alex

### Q\_\_\_\_\_\_\_\_\_\_\_\_\_

My understanding from the random forest is that it has bootstrapping built into it:  
<https://uk.mathworks.com/help/stats/treebagger-class.html>  
This is what is suggested for RF:

rng(1); % For reproducibility

Mdl = TreeBagger(50,meas,species,'OOBPrediction','On',...

'Method','classification')

Is the suggestion to bootstrap manually too (similar to logistic regression) in a loop and have the TreeBagger within that loop? and why would this double bootstrapping be necessary?

## Q 26/11

It's proving difficult to do this programmatically efficiently without this, because the function will throw an error if I were to supply a pair of parameter name and value that does not exist.  For example, the parameter 'PredictorSelection' has no 'default' value, so if I stick to CART (default), I must not supply an argument for this parameter.  Whereas if I choose 'curvature', then I would need to supply a parameter name and value pair. I.e.  
  
I.e., this would be fine:

cTree = fitctree(trnX, trnY, 'PredictorSelection', 'curvature');  
But not when I want to stay the default for PredictorSelection, this would throw an error.  
cTree = fitctree(trnX, trnY, 'PredictorSelection', 'default');

## A 25/11

Hi Behzad,

As for B bootstrap samples of the training set, to get the predictions of B base classifiers, you can set B = 10 (and then iterate, say, for i = 1:B) as set the number of bootstrap samples (and hence the number of base classifiers you will train, and hence the number of prediction sets that will be produced).

Both the random number generator seed and the number of bootstrap samples B can be considered to be hyperparameters of the bagged model. So, you can conduct a grid search over these two hyperparameters (e.g. B = [7, 10, 15], seed = [123, 567, 734]) to see if you can find an optimal combination of hyperparameter values.

## Q 24/11

Thank you for your response which has led me to revert to my "original" plan to use random forest.  
  
With regard to the resampling method, I understand from your perspective that bootstrapping should be used for fair comparison if one of the chosen methods is random forest. I appreciate that theoretically, though the technical implementation of bootstrapping for logistic regression took me 3 days.   
  
From initial analysis, the accuracy of logistic regression with bootstrapping is ~62% with ~0.5% fluctuation amongst 500 runs. This is ~13% lower than 5-fold cross-validation and about 100 times more computationally expensive. The poor performance of bootstrapped prediction is not in line with the literature; e.g. Mnich et al, 2020  
  
This discrepancy might primarily be due to the nature of my dataset as its heavily biased towards one class: for every death, there is 6 survival. Now I have these choices:  
  
1. To use a subset of data with equal representation of both classes. This can be backed up by literature, for example, Couronne et al, (2018) report that for logistic regression and random forest "results were noticeably dependent on the inclusion criteria used to select the example datasets".  
  
2. To use k-fold CV which might complicate comparison to random forest. I have, however, read papers with k-fold CV for comparing logistic regression (e.g. Alghamdi et al 2017) and random forest but unsure of the extent in which bootstrapping (a measure of reliability) can be fairly compared to k-fold CV (a measure of accuracy).   
  
3. To use an alternative second method which does not rely on bootstrapping; e.g. quadratic discriminant  
  
I would appreciate your input so that I can more forward. Many thanks.  
  
###########  
Mnich, K., Golińska, A.K., Polewko-Klim, A. and Rudnicki, W.R., 2020. Bootstrap Bias Corrected Cross Validation applied to Super Learning. arXiv preprint arXiv:2003.08342.  
  
Couronne et al, (2018): Couronné, R., Probst, P. and Boulesteix, A.L., 2018. Random forest versus logistic regression: a large-scale benchmark experiment. BMC bioinformatics, 19(1), p.270.  
  
Alghamdi, M., Al-Mallah, M., Keteyian, S., Brawner, C., Ehrman, J. and Sakr, S., 2017. Predicting diabetes mellitus using SMOTE and ensemble machine learning approach: The Henry Ford ExercIse Testing (FIT) project. PloS one, 12(7), p.e0179805.

## Q 20/11

I'm currently using GridSearch as one of the ways I'm tuning and optimising the hyperparameters for my Random Forest however I'm concerned with the total run-time it takes to derive the hyperparameters.

It is currently taking several hours to run and I'm worried it might not be feasible when submitting.

**Code:**

**rf = fitcensemble(X\_train, y\_train, "OptimizeHyperparameters","auto",'Learners',t, ...**

**'HyperparameterOptimizationOptions',struct('Optimizer','gridsearch'))**

Is there anything I could do to speed up the process? I'm not too sure it'd make sense submitting this as the person marking this cannot wait hours for the model to finish running.

## A 19/11

### A\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

You are right, 'Replace', false specifies sampling without replacement.

But we build trees on random samples with replacement. Subset sampling is used to make sure there is no structure to the dataset that is hidden by these strong predictor features.

### Q\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

I thought we bootstrapped WITH replacement and yet the code [~, idx] = datasample(y, 0.8\*numSamples, 2, 'Replace', false); doesn't replace. Is this correct?

# Decision Trees

## Q 25/11

I'm in the process of optimising my decision tree model, and want to confirm the optimization steps as well as asking some questions.

**Data Background:** My dataset is an imbalance binary classification problem (ratio of the true class is about 10% of the total data) about banking telemarketing campaign.  Since the nature of the true class is very small, I'm planning to use AUC and the area of cumulative gain curve (ALIFT) chart as indicators to measure the model performance (also aligned with the literature).

**Data Pre-processing:** I have split the data into training and test set. I also split the training set into 10 fold (using cvpartition)

**Concept of optimising Decision Tree:**

I'm planning to tune 3 hyperparameters; a maximum number of split, minimum leaf size, and tree pruning for this decision tree model.

1. Create a grid search to find the optimal (minimum value of ) maximum number of split and minimum leaf size that will give the best output for AUC and ALIFT Value when test with the validation set of a K-fold.

2. Pruning a tree on a fixed set of the parameter(maximum number of split and minimum leaf size) also use  AUC and ALIFT Value as an indicator

3. Select the best combination of these 3 parameters. Train the optimum model and test it against the test dataset.

Before I move on to further questions, I want to ask that is the above concept the right approach to optimise decision tree model? Any comment or advice is welcomed.

I already tried to implement the above approach and have encountered some conceptual problems.

**Question:**

1. What should be the upper limit for both the maximum numbers of split and leaf size when I do the initial set up.  
- For now, I trained one decision tree without specifying any parameters and take that number for that model depth to be my upper limit. As for the leaf size; I set my upper limit be half the amount(row) of my test data.

2. Which dataset should I test with when I'm prunning a tree? I did the tunning of the first two parameters using Kfold, but it seems to me that a model trained by KFold doesn't give the tree prune level. So should I training it will the whole training set and test the prunning against test set?

# Naïve Bayes Notes

## A 26/11

As for your bagged model, here are some optional suggestions. Yes, first by setting e.g. predTable = table, you can use the trained classifier to predict the class labels of the test set pred = predict(mdl, X\_test) and append the column of predictions to the results table predTable = [predTable pred].   
  
Then when producing the ensemble (final) predictions, you can initialise an empty cell array for the ensemble predictions, ensemblePred = {}, n = size(predTable,1).   
  
Then, use can consider using a loop, e.g.:  
for i = 1:n  
   predSet = table2cell(predTable(i,:))  
   predStat = tabulate(predSet)  
   statTable = cell2table(predStat) -- convert the prediction  
   [val, idx] = max(statTable.predStat2)  
   pred = char(statTable.predStat1(idx)) -- convert to character array  
   ensemblePred{i} = pred – and save the ensemble prediction  
end  
  
and reshape the ensemble predictions cell array to be a column vector rather than a row vector (i.e. so it matches Y\_train) n = size(ensemblePred, 2) and finalPred = reshape(ensemblePred, n, 1). Then, you can analyse the predictive performance, confusionmat(Y\_test, finalPred, ...) etc.  
  
Btw, there is also a way to save best model in compact mode, just FYI:  
  
opt\_mdl1 = TreeBagger(...)  
oobError1 = opt\_mdl1.oobError('Mode','Ensemble')  
save best model in compact mode:  
mdlC = opt\_mdl1.compact  
and use saved model for prediction on test set:  
[predClass,classifScore] = mdlC.predict(X\_test)  
  
Best,  
Alex

# Evaluation Notes

## A 24/11

Hi Phil,

You can compute true positive and false positive rate yourself, here are some additional tips on this:

Res = confusionmat(testingLabel,predLabel) -- confusion matrix, predLabel is derived from predict()

TruePos = Res(1,1)

FalsePos = Res(2,1)

FalseNeg = Res(1,2)

TrueNeg = Res(2,2)

Accuracy = (TruePos+TrueNeg)/(TruePos+TrueNeg+FalsePos+FalseNeg)

Precision = TruePos/(TruePos+FalsePos)

Recall = TruePos/(TruePos+FalseNeg)

F1\_Score = 2\*(Recall \* Precision) / (Recall + Precision)

and then plot a ROC curve with perfcurve().

Also, please check these links:

https://www.mathworks.com/help/deeplearning/ref/roc.html?searchHighlight=roc&s\_tid=srchtitle

https://www.mathworks.com/help/deeplearning/ref/plotroc.html

Hope that helps.

Best,

Alex

## Q 24/11

Also, I did look into plotroc and roc and they don't quite work. So, given I'll do it manually, is the approach of plotting cumulative FP versus cumulative FN the way to go?

## A 21/11

PCA is a part of exploratory data analysis and is used before building models (not randomly). Based on some relevant work, e.g. Andrew Y. Ng and Michael I. Jordan “On discriminative vs. generative classiﬁers: a comparison of Logistic Regression and Naive Bayes”, one can hypothesise that Naive Bayes is expected to reach its optimal generalisation error faster than logistic regression. In general, Naive Bayes performance is dependent on the tuning of hyper-parameters and feature engineering, whereas logistic regression is mainly dependent on feature engineering.

Checking the performance of a model with/without PCA will add richness to your analysis.

## Q 20/11

I've been working on the coursework and could you please give some advice on my question below?

**[Background Info]**

I built two ml models(Naive Bayes, Logistic Regression) and then applied a PCA classifier for each. Every classifier is having different value **k** for the reduced dimensionality.  
In general, the first model NB is faster for training then the second model LG. And the LG model is also finely optimised after applying a classifier.

**[Question]**

My initial hypothesis is, however, regardless of the value **k**, every NB should be faster than LG.

What I am questioning on my experiments is having a few **k** values which made NB performs better than LB. Is it possible to conclude that my classifier has a threshold number boosting the ability(speed) of the NB model? Or do I have to check my algorithms?

Every **k** are selected above 70% variance of principal components.

Thank you for your comments in advance.

## A 19/11

Dear Phil and all,

Please find attached optional notes on Random Forests to get extra insights on using Matlab's built-in functions and structuring your code. But feel free to follow your own approach.

Also, here is a simple example that explains information gain/entropy:

<https://victorzhou.com/blog/information-gain/>

## A 17/11

We're assuming a binary classification task with possible outcomes "positive" or "negative".

When you choose an attribute you split the data according to the possible values of that attribute. For each split (i.e. each branch of the tree if you were to make that choice of attribute) you want to calculate the proportion of positive examples (number of positive examples over the total number of examples in that split) as your estimate of the probability of that outcome (P(vi)). This is where pi/(pi+ ni) come from (for the i-th branch). Dually, you calculate ni/(pi+ ni) for the negative examples in that branch.

To calculate the information gain, you're looking to calculate H(positive,negative) for that choice of attribute. You do that by adding up the log probabilities for each branch. Attribute A which can take v  values divides E into subsets E1 … Ev each Ej having pj positive and nj negative examples, and you sum over j.

To finally choose the attribute to split on, you need to calculate the info gain of each possible attribute choice.

Best,  
Artur

## A 16/11

Another approach - Minimum-Redundancy Maximum Relevance method based on mutual information. Please check on MathWorks. Optionally a comparison can also be made between the latter and the simple mutual information ranking between each predictor and the target variable independently.

Here is the reference:

Peng, Hanchuan et al. “Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy.” IEEE Transactions on Pattern Analysis and Machine Intelligence 27 (2005): 1226-1238.

However, this method can be only applied for categorical variables (since it is more difficult to estimate mutual information between categorical and continuous variables), so probably one would need to discretize numeric features by thresholding them.

# Submit Notes

## A 20/11

PCA is a part of exploratory data analysis and is used before building models (not randomly). Based on some relevant work, e.g. Andrew Y. Ng and Michael I. Jordan “On discriminative vs. generative classiﬁers: a comparison of Logistic Regression and Naive Bayes”, one can hypothesise that Naive Bayes is expected to reach its optimal generalisation error faster than logistic regression. In general, Naive Bayes performance is dependent on the tuning of hyper-parameters and feature engineering, whereas logistic regression is mainly dependent on feature engineering.

Checking the performance of a model with/without PCA will add richness to your analysis.