Summative Assessment EMATM0061 - Part C

Akshat Pande | TX21857| 2153363 12/01/2022

Section C - Data Science Report

Theory

Supervised Learning - Random Forest

Definition- > Random forest is an ensemble of decision trees created through resampling and reducing entropy using different attributes and features from the data provided. Random forest performs prediction and plausible variable inferences through the majority of resampling through many decision trees.

Description

Key terminologies = - Entropy - measure of randomness in a dataset

$$tendsto0atprediction$$

- Information gain - change in entropy acheived by decreasing entropy at every decision level - Leaf node - carrier decision - Decision node - final decision (entropy tends to 0) - Root node - First node of the decision trees in the random forest (entropy is maximum)

Random forest steps - Create a bootstrapped dataset with a subset of available variables - Fit the decision trees given the dataset - Repeat many times with different attributes and bootstraps - Tally results - Class with most positive results or votes, are predicted as the truth

- Appropriate Data

Random forest are less influenced by outliers and can be used in datasets where outliers affect the outcome of the predictions Random forest classifier and regression is good at handling missing data and can also handle high variance (new data) without sharp decrease in accuracy, and that improves it predictive capabilities in such occasion. Random forest avoids overfitting It only utilizes the best features for prediction and wont use the extra features that may exist in the dataset, moreover, random forest can be used to chart variable inferences based on its importance.

Suitable dataset for training

- pre-requisities
- validation
- test/split

Data Validation

Importing the dataset

dataset <- read.csv('Data.csv') #Reads the dataset into a data frame

Taking care of missing data

Part 1 checks if NA values exist and then replaced them with the mean of the column

dataset_column1<- ifelse(is.na(dataset_column1), ave(dataset_column1, FUN = function(x) mean(x, na.rm = TRUE)), dataset_column1) dataset_column2 <- ifelse(is.na(dataset_column2), ave(dataset_column2, FUN = function(x) mean(x, na.rm = TRUE)), dataset_column2)

Encoding categorical data into numerical values for easy processing without dropping.

dataset_column3 <- factor(dataset_column3, levels <- c('A', 'B', 'C'), labels <-c(1, 2, 3)) dataset_column4 <- factor(dataset_column4, levels<-c('No', 'Yes'), labels <- c(0, 1)) * Replace _ with $

Changedduetoexpressionsbeingimplementedinrmd

Data Preprocessing and Split

Importing the dataset

dataset<- read.csv('Data.csv')

Splitting the dataset into the Training set and Test set

install.packages('caTools') library for splitting

library(caTools) set.seed(123) split <-sample.split(dataset$DependentVariable, SplitRatio = 0.8) training_set <- subset(dataset, split == TRUE) test_set <-subset(dataset, split == FALSE)

Feature Scaling

training_set <- scale(training_set) test_set<-scale(test_set)

Implementation

Code Example

```
#Sourcecode : https://github.com/matbenni/DigitRecognizerInR/blob/master/Digit_Recognizer_Code.R

library(lattice)
library(ggplot2)
library(caret)
library(randomForest)
```

```
## randomForest 4.6-14

## Type rfNews() to see new features/changes/bug fixes.

##
## Attaching package: 'randomForest'

## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
#Readign test and train datasets into dataframes
digit.test <- read.csv("test.csv", header = TRUE)
digit.train <- read.csv("train.csv", header = TRUE)
dim(digit.test)
```

```
## [1] 28000   784
```

```
dim(digit.train)
```

```
## [1] 42000   785
```

```
digit.train$label <- as.factor(digit.train$label)

str(digit.train)
```

```
## 'data.frame':    42000 obs. of  785 variables:
##  $ label   : Factor w/ 10 levels "0","1","2","3",..: 2 1 2 5 1 1 8 4 6 4 ...
##  $ pixel0  : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel1  : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel2  : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel3  : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel4  : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel5  : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel6  : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel7  : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel8  : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel9  : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel10 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel11 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel12 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel13 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel14 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel15 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel16 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel17 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel18 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel19 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel20 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel21 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel22 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel23 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel24 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel25 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel26 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel27 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel28 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel29 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel30 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel31 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel32 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel33 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel34 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel35 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel36 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel37 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel38 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel39 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel40 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel41 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel42 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel43 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel44 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel45 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel46 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel47 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel48 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel49 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel50 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel51 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel52 : int  0 0 0 0 0 0 0 0 0 0 ...
```

```
##  $ pixel53 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel54 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel55 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel56 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel57 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel58 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel59 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel60 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel61 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel62 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel63 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel64 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel65 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel66 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel67 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel68 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel69 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel70 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel71 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel72 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel73 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel74 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel75 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel76 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel77 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel78 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel79 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel80 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel81 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel82 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel83 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel84 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel85 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel86 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel87 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel88 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel89 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel90 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel91 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel92 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel93 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel94 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel95 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel96 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel97 : int  0 0 0 0 0 0 0 0 0 0 ...
##   [list output truncated]
```

```
table(digit.train$label)
```

```
##
##    0    1    2    3    4    5    6    7    8    9
## 4132 4684 4177 4351 4072 3795 4137 4401 4063 4188
```

```
prop.table(table(digit.train$label))*100
```

```
##
##         0         1         2         3         4         5         6         7
##  9.838095 11.152381  9.945238 10.359524  9.695238  9.035714  9.850000 10.478571
##         8         9
##  9.673810  9.971429
```

```
#randomforest dataframe that stores the results of randomforest classifier prediction
s.
#ntree
randomforest1 <- randomForest(label ~ ., data = digit.train, ntree = 50)

pred1 <- predict(randomforest1, digit.test)

output <- data.frame(pred1)

output.submit = data.frame(ImageId = seq(1,length(output$pred1)), Label = output$pred
1)

# Write the solution to file
write.csv(output.submit, file = 'Solution.csv', row.names = F)
```

Describe Dataset

The dataset example taken above contains pixel values for training and testing for different images. It denotes that all the images are of the same size and the pixel values show if there exists any pixel there, by comparing it with pixel occurence in usual digits and performing a correlation statistics using decision trees randomforest can easily find out the digit type

Appropriate metric for performance

Training data variation

Varying training data in Random forest can lead to different results, usually while Random forest is good at handling high variance or new data, the kind of data that exists in the dataset can affect its results, if there is a categorical data that is present in numerous level of decisions trees or/and the data is highly correlated then the predicition through Random forest is not accurate.

Cross Validation

Cross Validation or CV by splitting existing data into smaller segments using nfolds or kfolds and performing the same analysis on them and comparing the results is not as applicable to Random forest as other machine learning algorithm. Primarily, as it already works as an ensemble for decision trees and randomizes the entropy reduction randomly, which already leads to reduction of data, above so, it works on inference of variable importance and applying CV to the mix does not make much of a difference, similar to cross validation, increasing the value of ntree may affect the prediction but does not lead to substantial increase or decrease in overfitting.

Conclusion

Random forest is a good classification and regression algorithm that can be applicable to both supervised and unsupervised learning techniques, its strength exists in an indepth evolutionary analysis over the decision tree algorithm, moreover it can inference important features/variables can be used as an intermediatory to simplify complex algorithms(eg neural network), with fewer variables to consider. Random forest tends to be slow as its an ensemble of Decision trees, which basically means that it runs multiple decision trees algorithms consecutively and has an affect on memory consumption

References

Source code - Github and Kaggle - Digit Recognizer
https://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm (https://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm) - Random Forests Leo Breiman and Adele Cutler

https://hastie.su.domains/ElemStatLearn/printings/ESLII_print12_toc.pdf (https://hastie.su.domains/ElemStatLearn/printings/ESLII_print12_toc.pdf) - Elements of Statistical Learning