Predicting Star Type

Rajdeep Brahma [1] Pratik Lakhani[2] Diptanil Santra [3]
Piyush Swarnkar[4]

Data Set

The data set consisting of several features of stars i.e

- Absolute Temperature (in K)
- Relative Luminosity (L/Lo)
- Relative Radius (R/Ro)
- Absolute Magnitude (Mv)
- Star Color (white,Red,Blue,Yellow,yellow-orange etc)
- Spectral Class (O,B,A,F,G,K,,M)
- Star Type (Red Dwarf, Brown Dwarf, White Dwarf, Main Sequence , SuperGiants, HyperGiants)

The purpose of making the dataset is to prove that the stars follow a certain graph in the celestial Space , specifically called Hertzsprung-Russell Diagram or simply HR-Diagram so that we can classify stars by plotting its features based on that graph.

Data Collection and Processing

The dataset is created based on several equations in astrophysics.It has data for 240 stars which are mostly collected from web.The missing data were manually calculated using equations of astrophysics.
The Star Colour summary of the given data was :

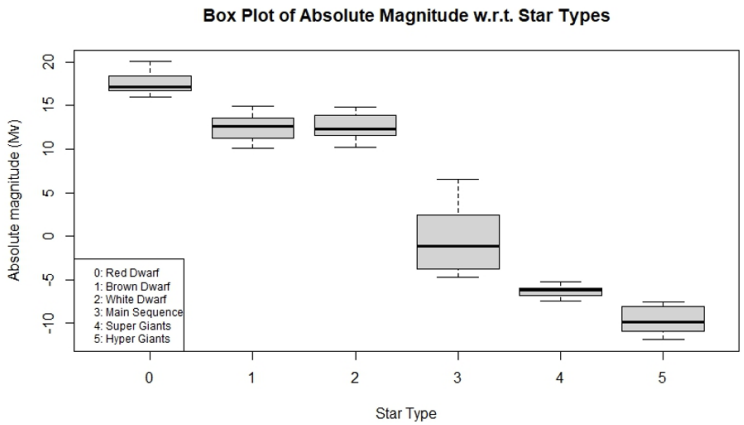
Blue	Blue	Blue white	Blue White	Blue white
55	1	3	10	1
Blue-white	Blue-White	Orange	Orange-Red	Pale yellow orange
26	1	2	1	1
Red	white	White	White-Yellow	Whitish
112	3	7	1	2
yellow-white	yellowish	Yellowish	Yellowish White	
8	2	1	3	

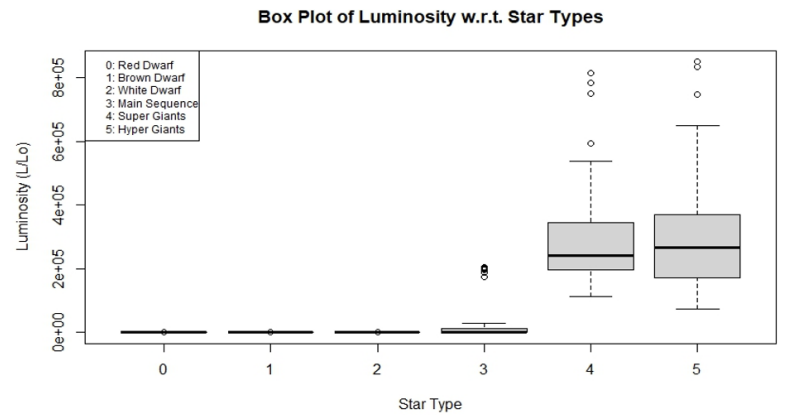
So the data needed some cleaning and the final summary was:

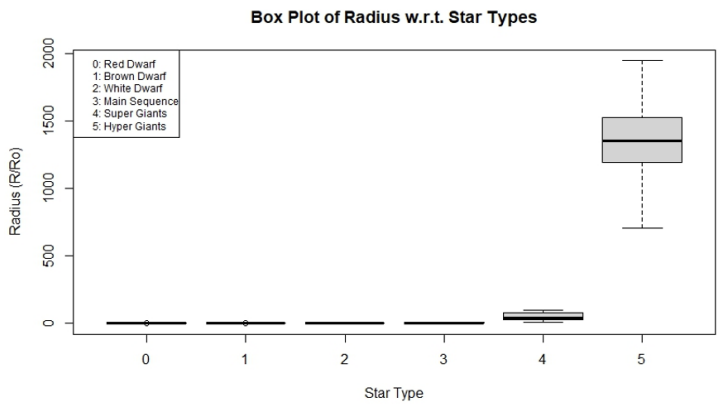
Blue	Blue-white	Orange	Orange-Red
56	41	2	1
Pale yellow orange	Red	White	Whitish
1	112	10	2
yellow-white	yellowish		
12	3		

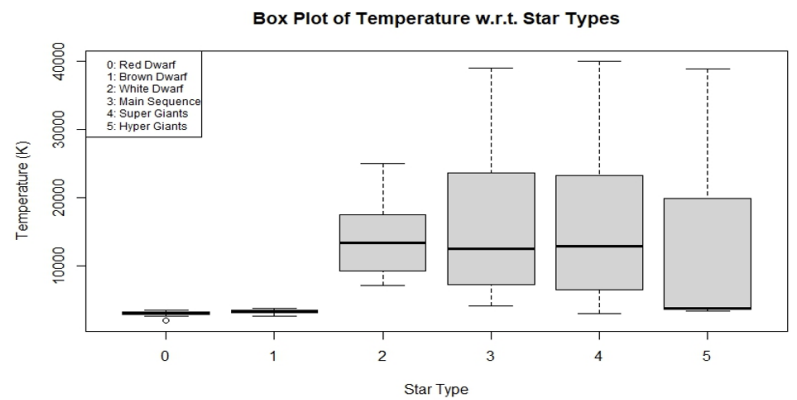
Preliminary Analysis:Box Plot

Here we have attached box plots of star type with different predictors.



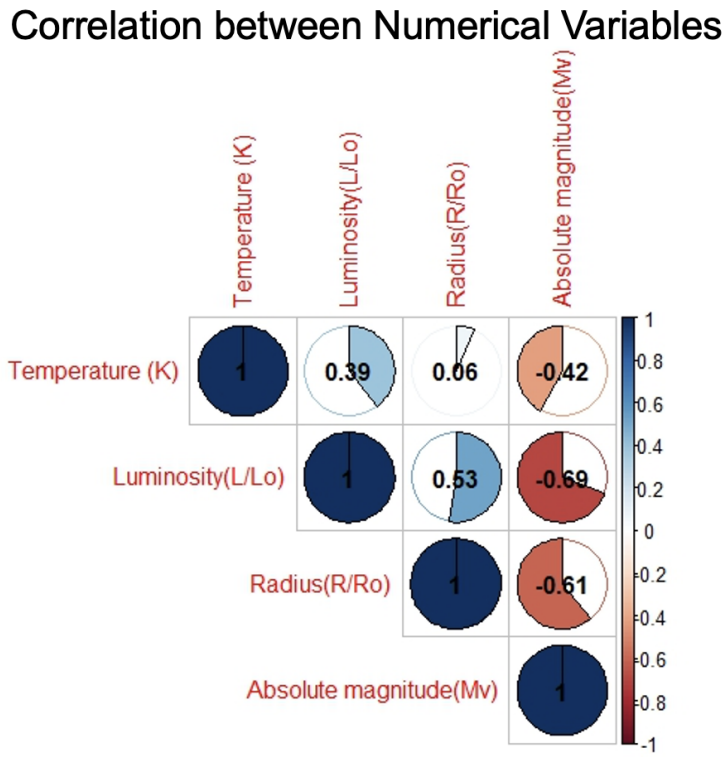






Preliminary Analysis:Correlation among predictors

Here we have attached the correlation among different predictors.



Methodology

Out of all the predictors we have,Spectral Class and Star color are factors.One way to tackle it is we will carry out our analysis ignoring those two variables.
The exclusion can be justified by the fact that theoretical background-information pertaining to the problem, i.e., Astrophysics/Astronomy firmly indicates that the information on Star Color and Star Spectral Class are functions of the "Star temperature".

Spectral Type	Surface Temperature	Distinguishing Features
O	> 25,000K	H; HeI; HeII
B	10,000-25,000K	H; HeI; HeII absent
A	7,500-10,000K	H; CaII; HeI and HeII absent
F	6,000-7,500K	H; metals (CaII, Fe, etc)
G	5,000-6,000K	H; metals; some molecular species
K	3,500-5,000K	metals; some molecular species
M	< 3,500K	metals; molecular species (TiO!)
C	< 3,500K	metals; molecular species (C2!)

Proper Splitting to Training/Testing Data

```
> table(data.play$Col)

             blue         blue white             orange         orange red
               56                 41                  2                  1
pale yellow orange                red              white       white yellow
                1                112                 10                  1
          whitish       yellow white          yellowish    yellowish white
                2                  8                  3                  3
> table(data.play$Spec)

  A   B   F   G   K   M   O
 19  46  17   1   6 111  40
> table(data.play$Type)

 0  1  2  3  4  5
40 40 40 40 40 40
```

The full dataset of 240 observations has various levels for each of the three categorical variables; these levels intersect with each other

```
> data.train=stratified(data.play,group=c("Type","Spec","Col"),0.80,bothSets = T)
  data.test=as.data.frame(data.train[[2]])
  data.train=as.data.frame(data.train[[1]])
```

Roughly 80% of the data is taken as training data

```
> table(data.train$Col)

             blue         blue white             orange         orange red
               45                 33                  2                  1
pale yellow orange                red              white       white yellow
                1                 90                  8                  1
          whitish       yellow white          yellowish    yellowish white
                2                  6                  2                  2
> table(data.train$Spec)

  A   B   F   G   K   M   O
 15  38  13   1   5  89  32
> table(data.train$Type)

 0  1  2  3  4  5
32 32 31 32 32 34
```

The split has been made in such a way that the training dataset has at least one observation from each level of each of the three categorical variables; proportions are as close to 0.8, as possible, for each level (w.r.t. full data)

R Package(s) used:
->splitstackshape: has the useful stratified() function

Some Domain Knowledge

->Theoretical background-information pertaining to the problem, i.e., Astrophysics/Astronomy firmly indicates that the information on "**Star Color**" and "**Star Spectral Class**" are functions of the "**Star temperature**".

->Following websites substantiate the above mentioned statements:

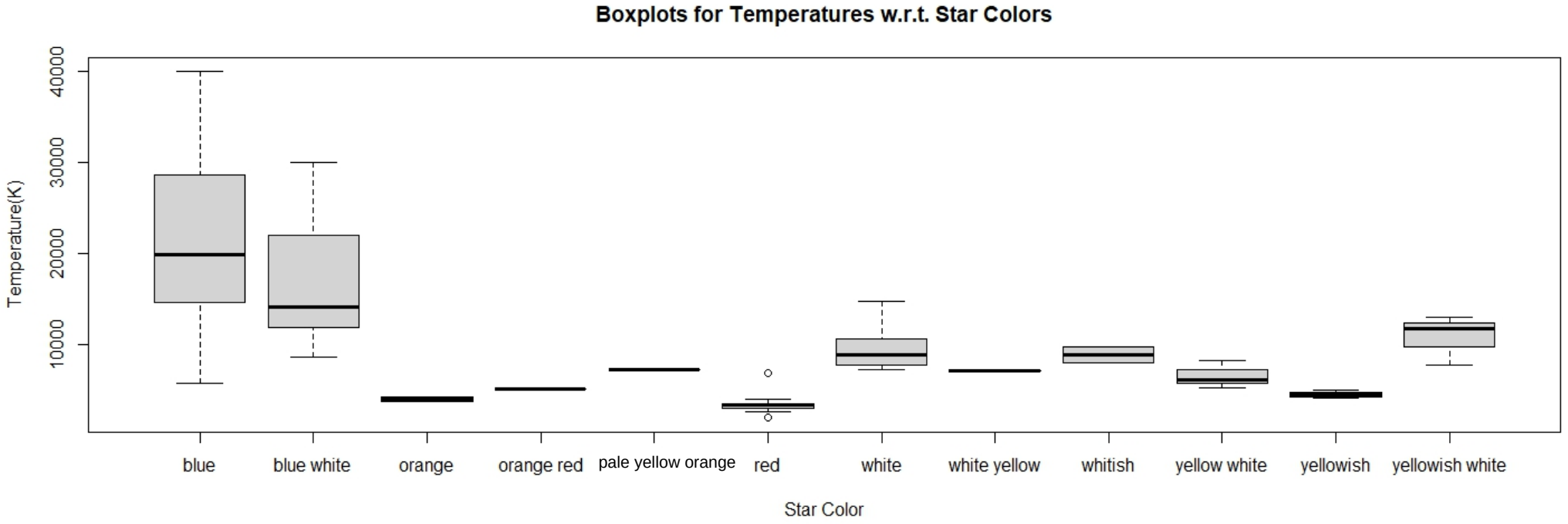
-->*On relationship between Star Colour and Star Temperature*

https://lco.global/spacebook/distance/magnitude-and-color/#:~:text=The%20surface%20temperature%20of%20a%20star%20determines%20the%20color%20of,are%20hotter%20than%20red%20stars

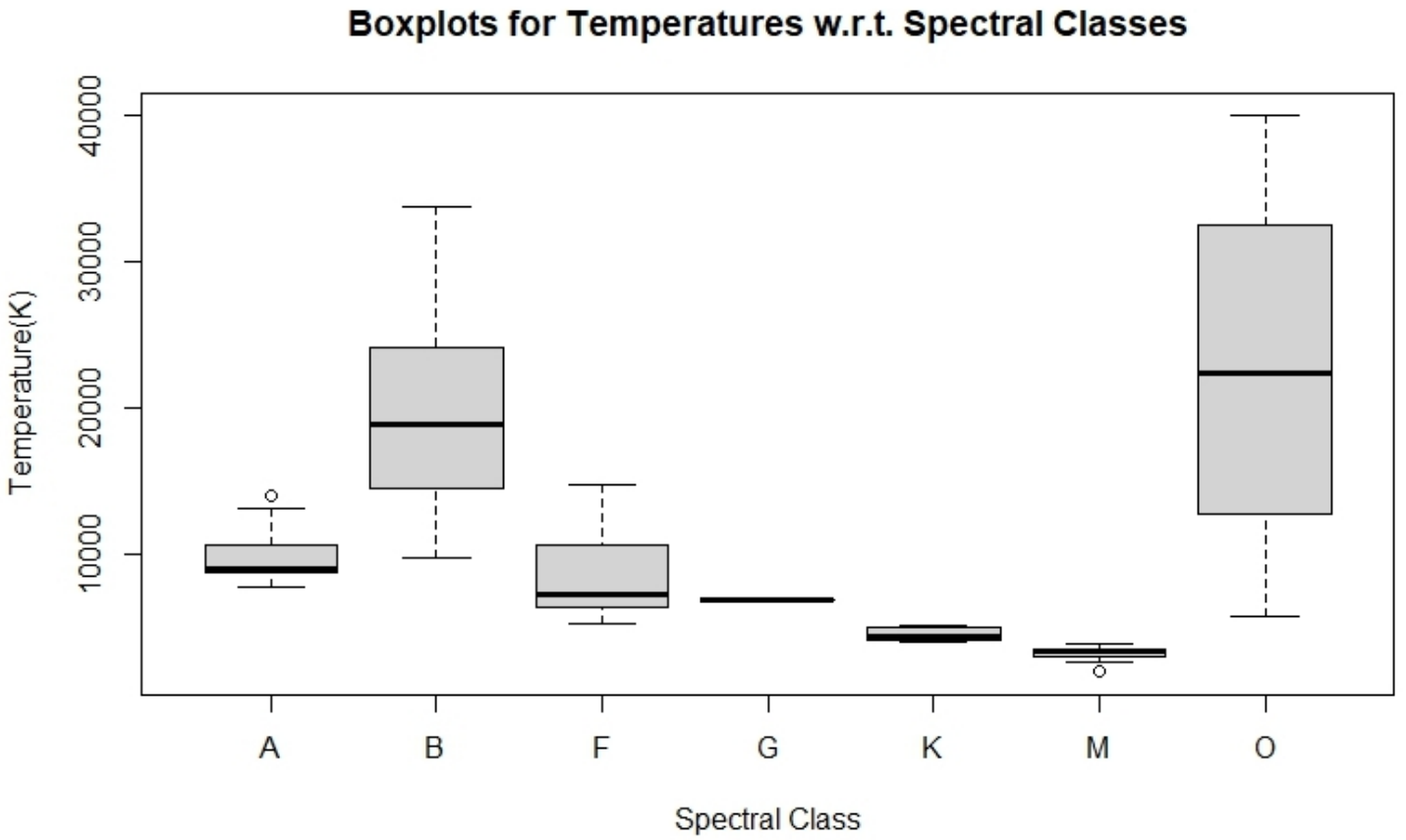
-->*On relationship between Star Spectral Class and Star Temperature*

https://lweb.cfa.harvard.edu/~pberlind/atlas/htmls/note.html#:~:text=Each%20spectral%20type%20is%20divided,%22color%22%20and%20surface%20brightness

Some Empirical Verification of Domain Knowledge



Some Empirical Verification of Domain Knowledge



Decision Tree(s) and Random Forest(s)

Decision Tree(s)

One Decision Tree based on all 6 variables

```
Call:
 randomForest(formula = Type ~ ., data = data.train, mtry = 4,      importance = TRUE, ntree = 1)
               Type of random forest: classification
                     Number of trees: 1
No. of variables tried at each split: 4

        OOB estimate of  error rate: 2.99%
Confusion matrix:
  0  1  2 3  4  5 class.error
0 8  0  0 0  0  0       0.000
1 0 10  0 0  0  0       0.000
2 0  0 15 0  0  0       0.000
3 0  0  0 7  0  0       0.000
4 0  0  0 0 11  0       0.000
5 0  0  0 0  2 14       0.125
>
> bag.train = predict(bag.class ,newdata = data.train[,c(-7)])
> cmatrix = table(bag.train ,data.train$Type)
> cmatrix

bag.train  0  1  2  3  4  5
        0 32  0  0  0  0  0
        1  0 32  0  0  0  0
        2  0  0 31  0  0  0
        3  0  0  0 32  0  0
        4  0  0  0  0 32  2
        5  0  0  0  0  0 32
> sum(diag(cmatrix))/sum(cmatrix)#accuracy
[1] 0.9896373
>
> bag.test = predict(bag.class,newdata=data.test[,-7])
> cmatrix = table(bag.test ,data.test$Type)
> cmatrix

bag.test 0 1 2 3 4 5
       0 8 0 0 0 0 0
       1 0 8 0 0 0 0
       2 0 0 9 1 0 0
       3 0 0 0 7 0 0
       4 0 0 0 0 8 0
       5 0 0 0 0 0 6
> sum(diag(cmatrix))/sum(cmatrix)#accuracy
[1] 0.9787234
```

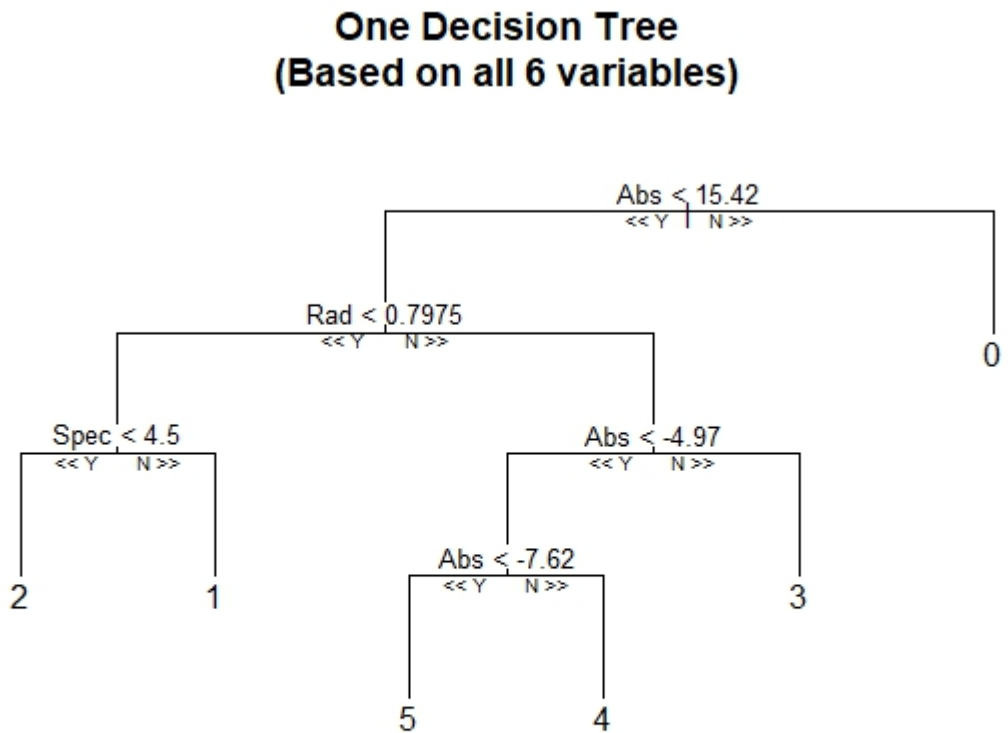
Description of the decision tree (which is equivalent to a random forest with just one tree)

Evaluating the decision tree on training data

Evaluating the decision tree on testing data

R Package(s) used:
->randomForest: to build the decision-tree(s)
->reprtree: to plot the decision-tree(s)

Decision Tree(s)



R Package(s) used:
->randomForest: to build the decision-tree(s)
->reprtree: to plot the decision-tree(s)

Decision Tree(s)

One Decision Tree based on only 4 numerical variables

```
Call:
 randomForest(formula = Type ~ ., data = data.train[, c(-5, -6)],      mtry = 4, importance = TRUE, ntree
 = 1)
               Type of random forest: classification
                     Number of trees: 1
No. of variables tried at each split: 4

        OOB estimate of  error rate: 1.33%
Confusion matrix:
   0  1  2  3  4  5 class.error
0 10  0  0  0  0  0  0.00000000
1  0 12  0  0  0  0  0.00000000
2  0  0 11  0  0  0  0.00000000
3  0  0  0 12  0  0  0.00000000
4  0  0  0  1 13  0  0.07142857
5  0  0  0  0  0 16  0.00000000
>
> bag.train = predict(bag.class ,newdata = data.train[,c(-5,-6,-7)])
> cmatrix = table(bag.train ,data.train$Type)
> cmatrix

bag.train  0  1  2  3  4  5
        0 32  0  0  0  0  0
        1  0 32  0  0  0  0
        2  0  0 31  0  0  0
        3  0  0  0 32  1  0
        4  0  0  0  0 31  0
        5  0  0  0  0  0 34
> sum(diag(cmatrix))/sum(cmatrix)#accuracy
[1] 0.9948187
>
> bag.test = predict(bag.class,newdata=data.test[,c(-5,-6,-7)])
> cmatrix = table(bag.test ,data.test$Type)
> cmatrix

bag.test 0 1 2 3 4 5
       0 8 0 0 0 0 0
       1 0 8 0 0 0 0
       2 0 0 9 0 0 0
       3 0 0 0 8 1 0
       4 0 0 0 0 7 0
       5 0 0 0 0 0 6
> sum(diag(cmatrix))/sum(cmatrix)#accuracy
[1] 0.9787234
```

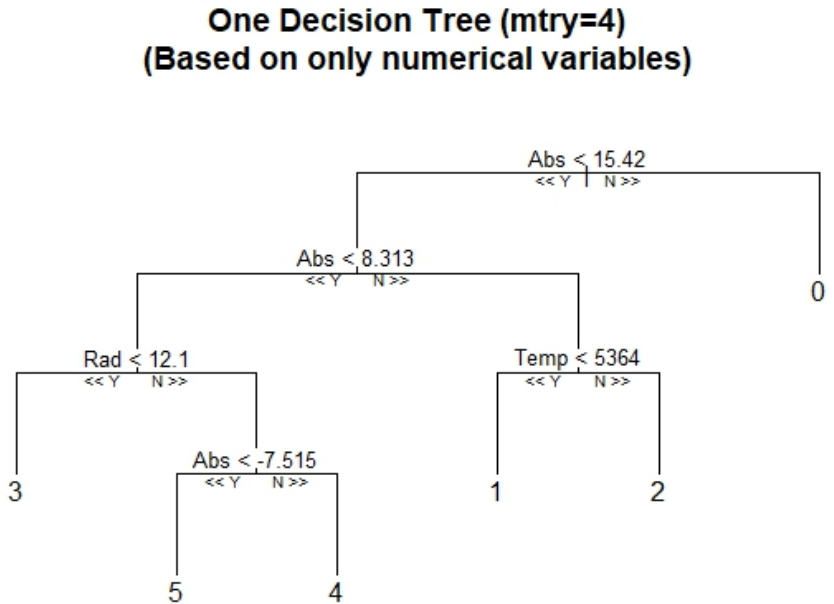
Description of the decision tree (which is equivalent to a random forest with just one tree)

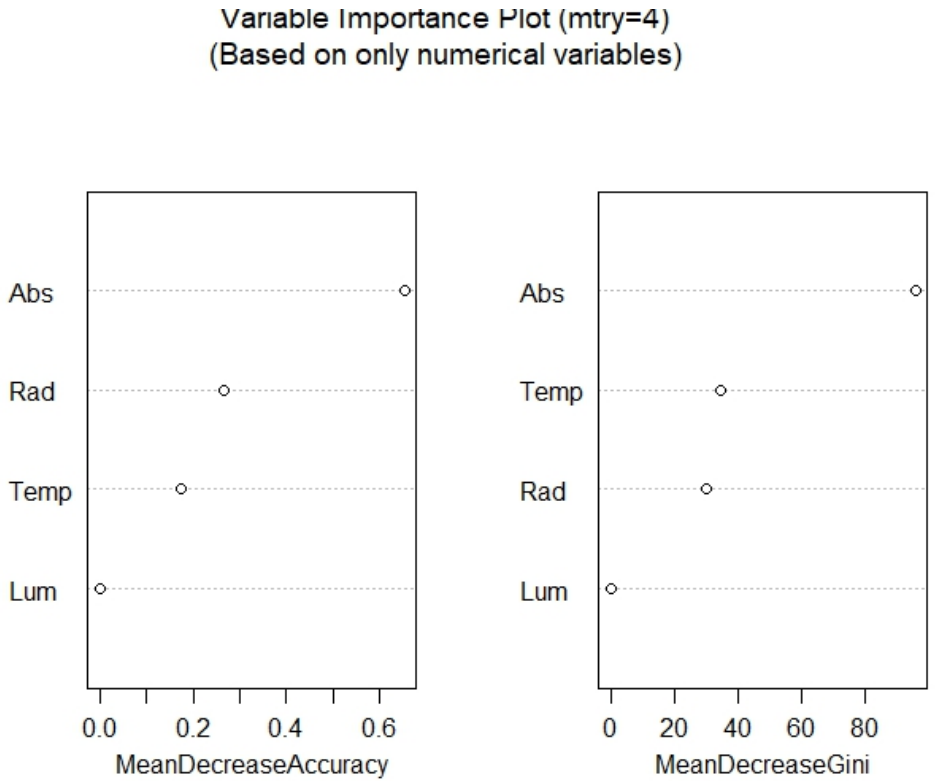
Evaluating the decision tree on training data

Evaluating the decision tree on testing data

R Package(s) used:
->randomForest: to build the decision-tree(s)
->reprtree: to plot the decision-tree(s)

Decision Tree(s)

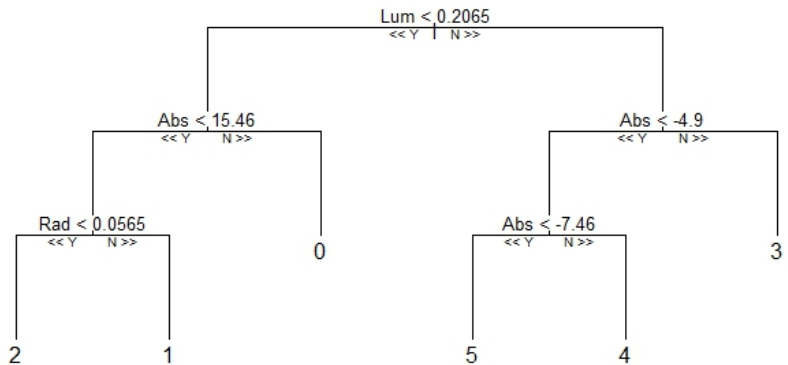




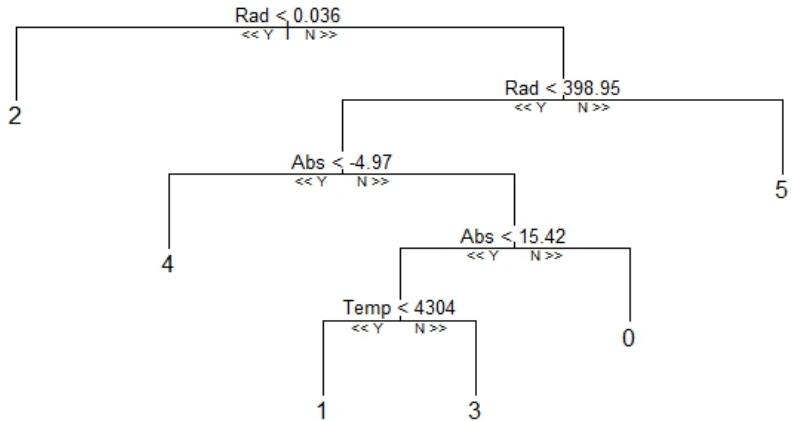
R Package(s) used:
->randomForest: to build the decision-tree(s)
->reprtree: to plot the decision-tree(s)

Random Forest(s)

An ensemble oriented Decision Tree (mtry=2, only numerical variables)
(For a random forest with 10 decision trees)

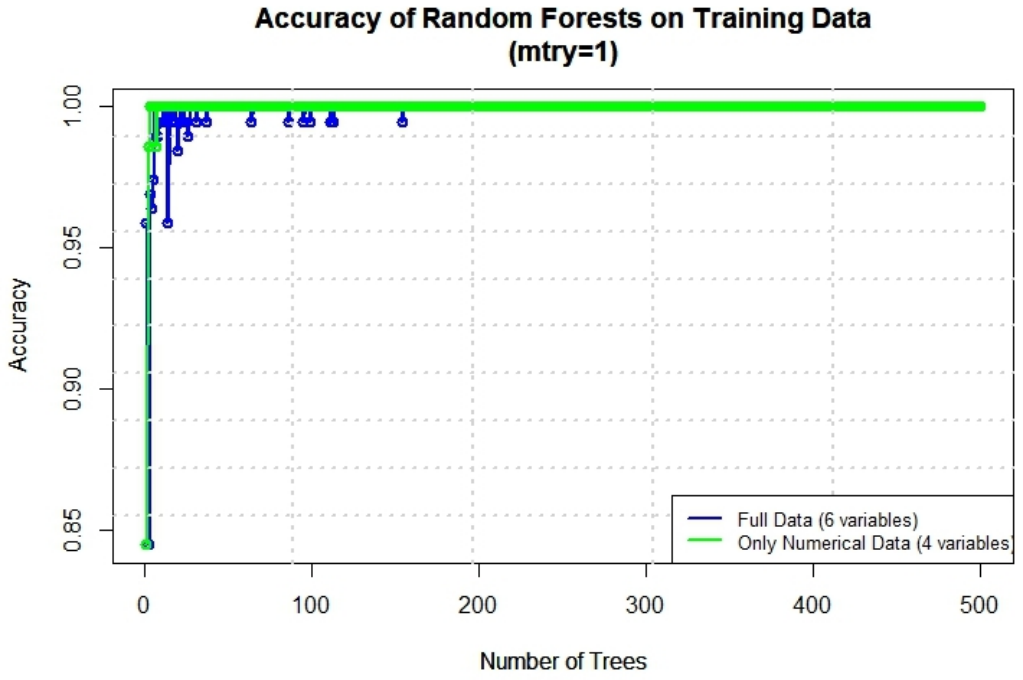


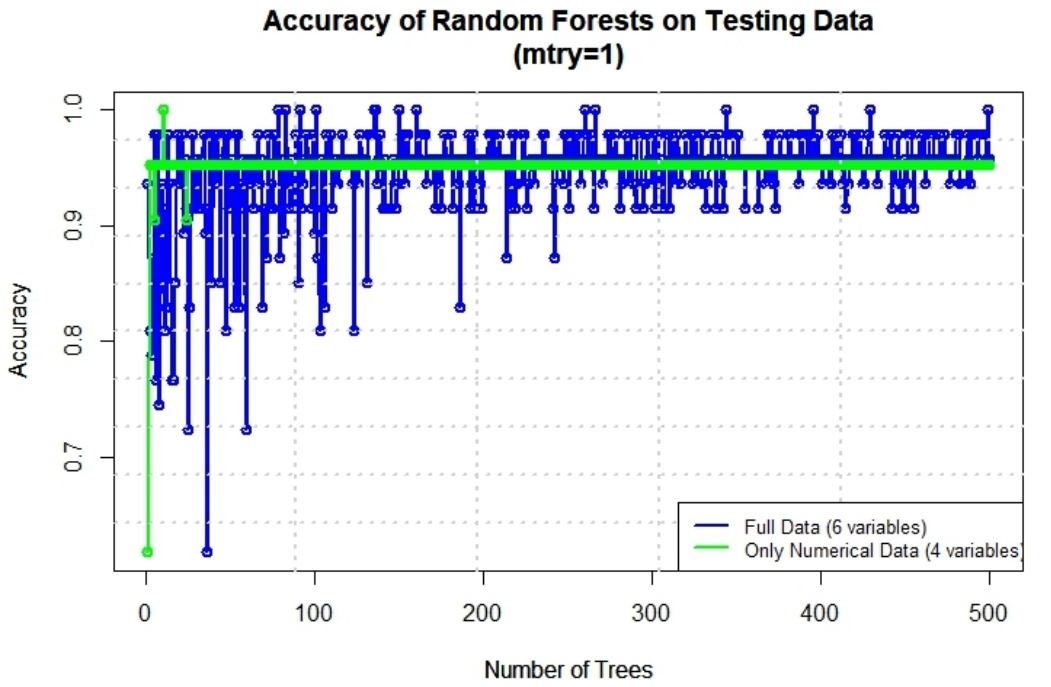
An ensemble oriented Decision Tree (mtry=4)
(For a random forest, based on all variables, with 10 decision trees)



R Package(s) used:
->randomForest: to build the decision-tree(s)
->reprtree: to plot the decision-tree(s)

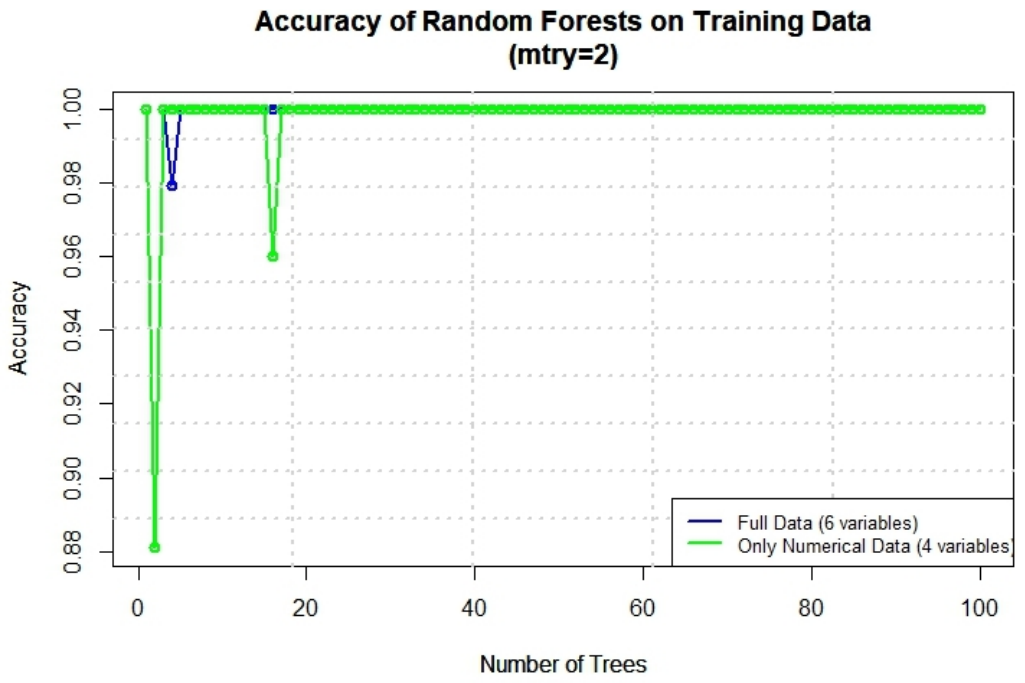
Random Forest(s)

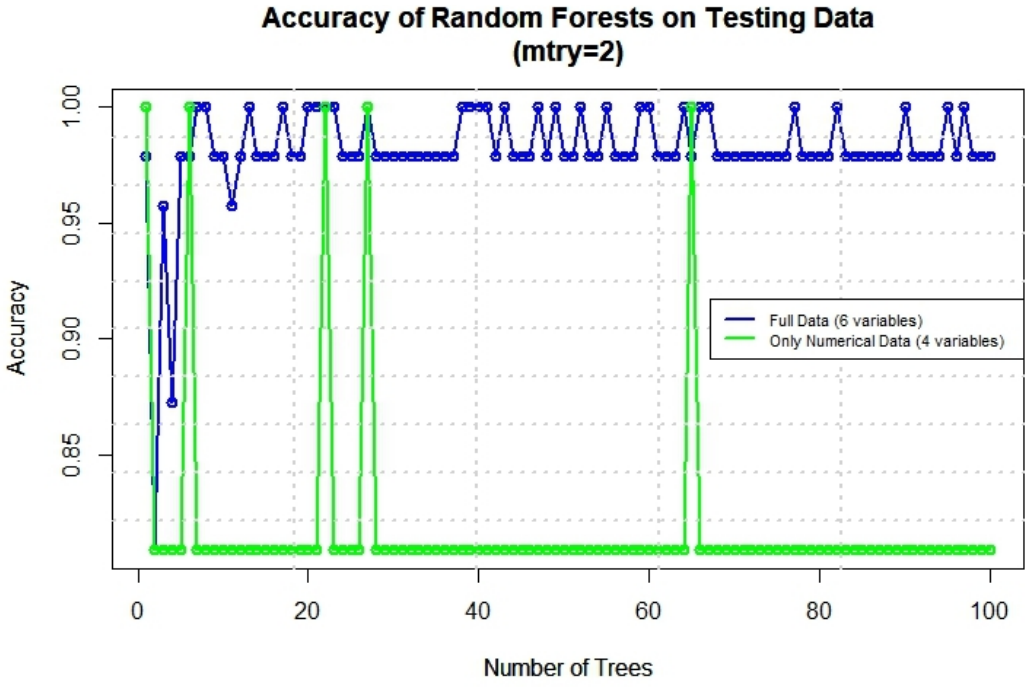




500 different random forests were created, each having different/unique number of trees ranging from 1 to 500
Number of variables considered while splitting a node is 1

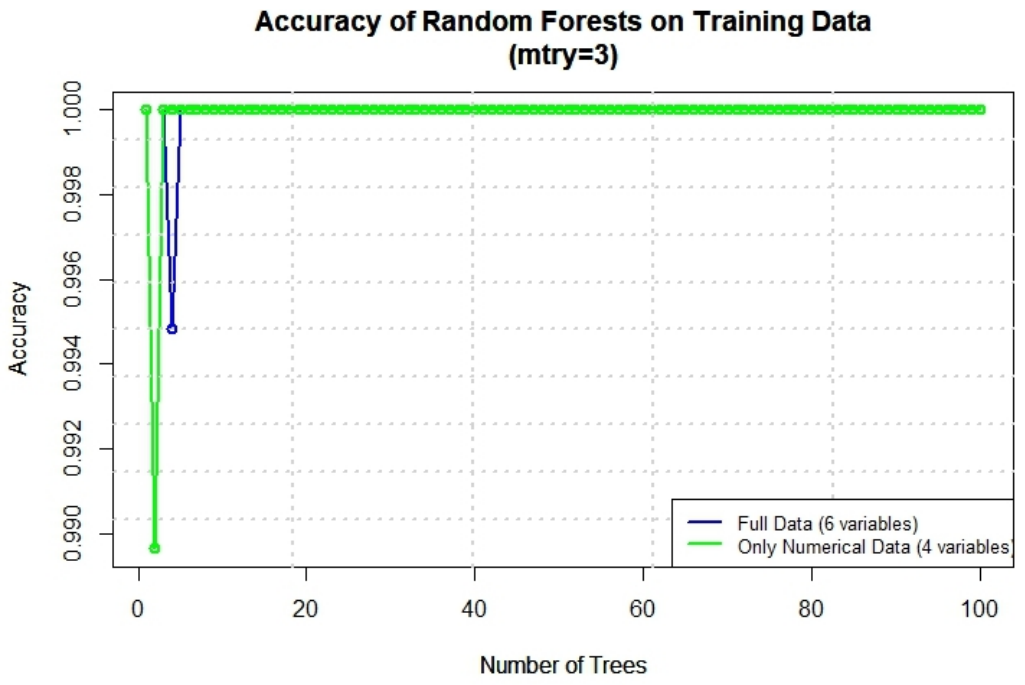
Random Forest(s)

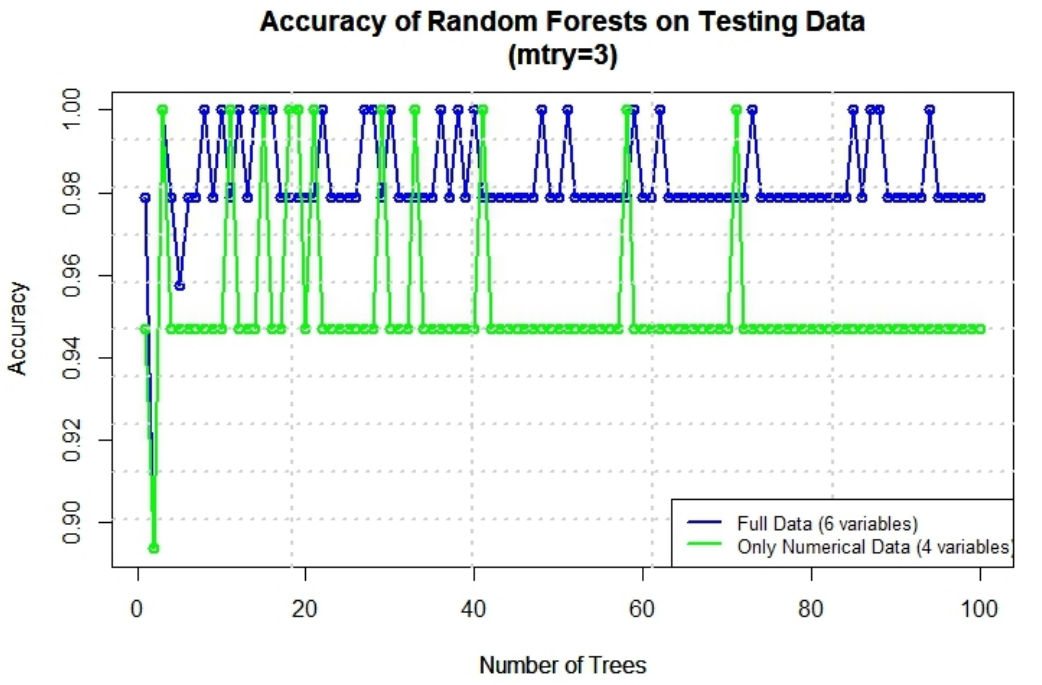




100 different random forests were created, each having different/unique number of trees ranging from 1 to 100
Number of variables considered while splitting a node is 2

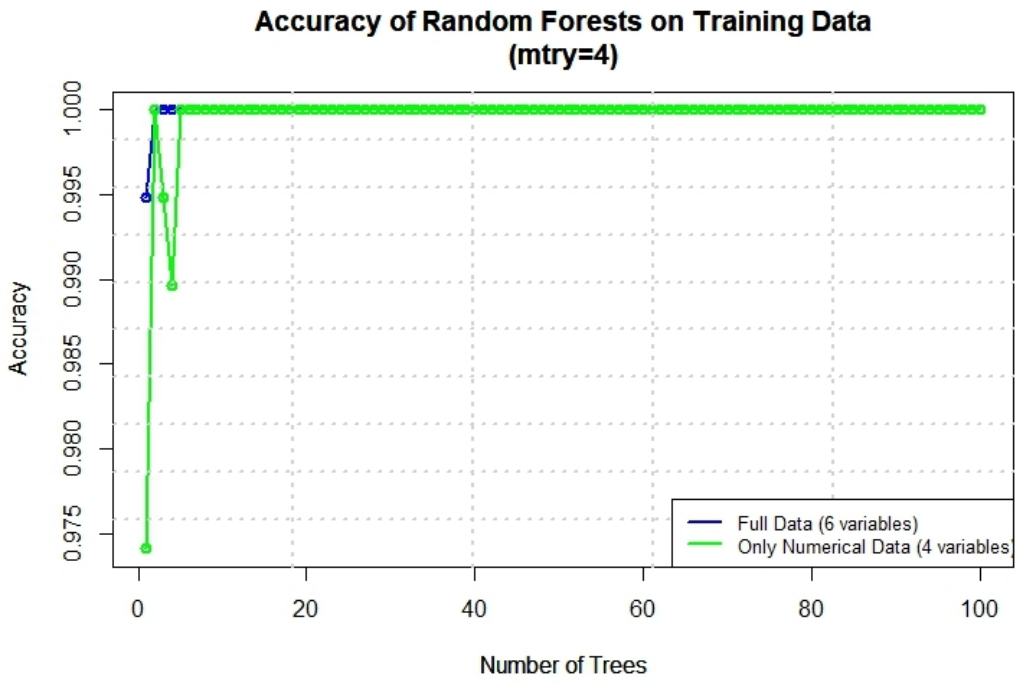
Random Forest(s)

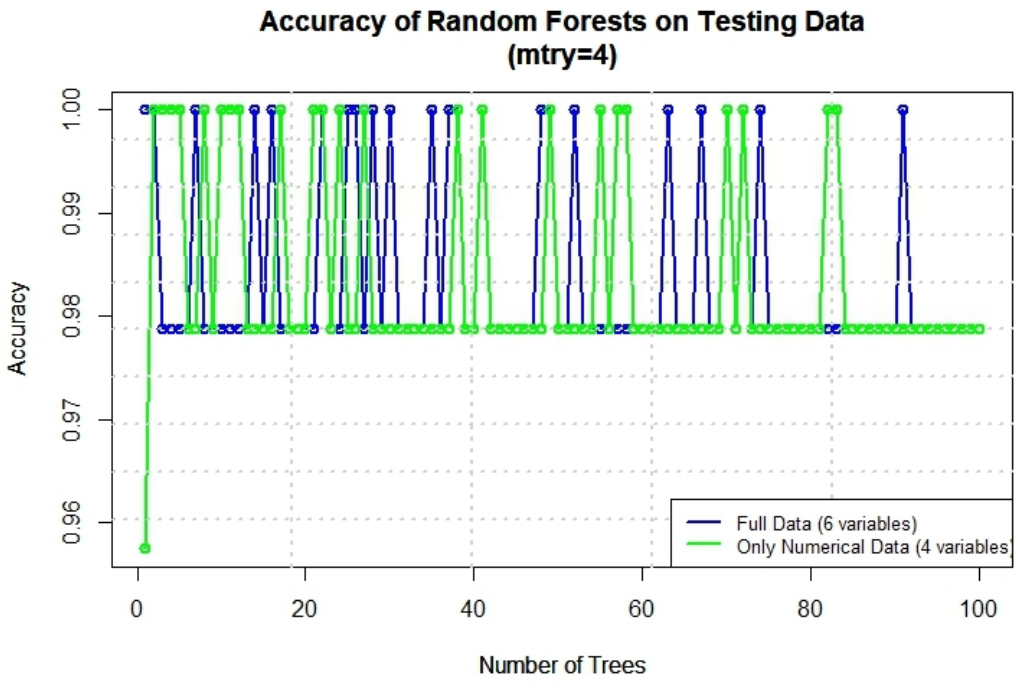




100 different random forests were created, each having different/unique number of trees ranging from 1 to 100
Number of variables considered while splitting a node is 3

Random Forest(s)



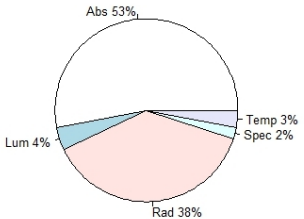


100 different random forests were created, each having different/unique number of trees ranging from 1 to 100
Number of variables considered while splitting a node is 4

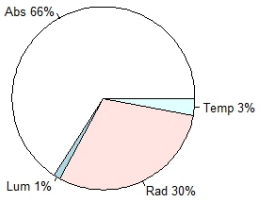
Random Forest(s)

Most Important Variable (as per voting)

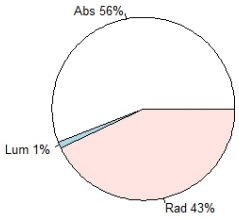
Most Important Variable (mtry=1)
w.r.t. Mean Decrease in Accuracy (using all variables)



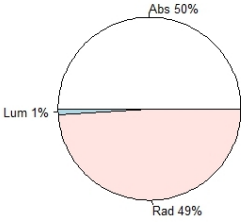
Most Important Variable (mtry=1)
w.r.t. Mean Decrease in Accuracy (using only numerical variables)



Most Important Variable (mtry=1)
w.r.t. Mean Decrease in Gini-Coefficient (using all variables)

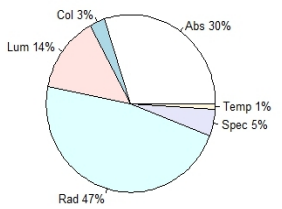


Most Important Variable (mtry=1)
w.r.t. Mean Decrease in Gini-Coefficient (using only numerical variables)

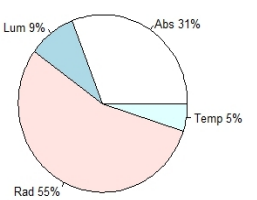


2nd-Most Important Variable (as per voting)

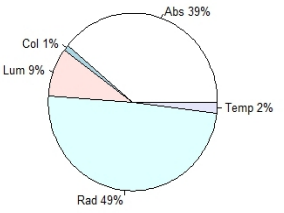
2nd-Most Important Variable (mtry=1)
w.r.t. Mean Decrease in Accuracy (using all variables)



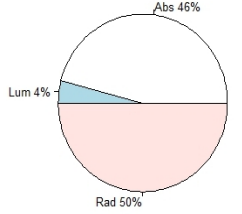
2nd-Most Important Variable (mtry=1)
w.r.t. Mean Decrease in Accuracy (using only numerical variables)



2nd-Most Important Variable (mtry=1)
w.r.t. Mean Decrease in Gini-Coefficient (using all variables)



2nd-Most Important Variable (mtry=1)
w.r.t. Mean Decrease in Gini-Coefficient (using only numerical variables)

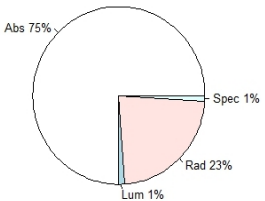


Number of variables considered while splitting a node is 1

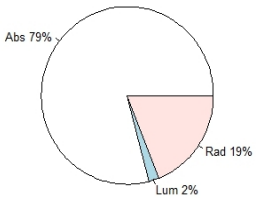
Random Forest(s)

Most Important Variable (as per voting)

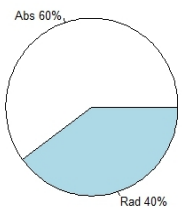
Most Important Variable (mtry=2)
w.r.t. Mean Decrease in Accuracy (using all variables)



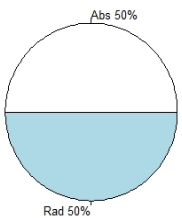
Most Important Variable (mtry=2)
w.r.t. Mean Decrease in Accuracy (using only numerical variables)



Most Important Variable (mtry=2)
w.r.t. Mean Decrease in Gini-Coefficient (using all variables)

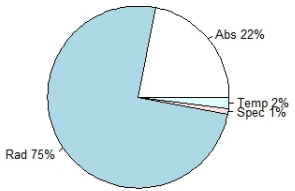


Most Important Variable (mtry=2)
w.r.t. Mean Decrease in Gini-Coefficient (using only numerical variables)

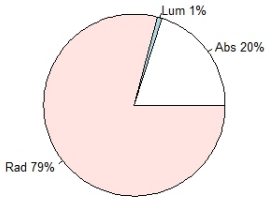


2nd-Most Important Variable (as per voting)

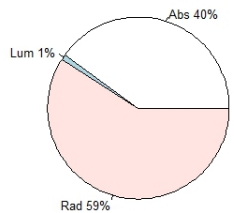
2nd-Most Important Variable (mtry=2)
w.r.t. Mean Decrease in Accuracy (using all variables)



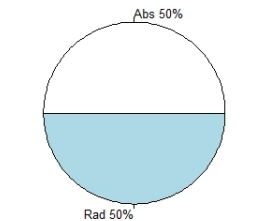
2nd-Most Important Variable (mtry=2)
w.r.t. Mean Decrease in Accuracy (using only numerical variables)



2nd-Most Important Variable (mtry=2)
w.r.t. Mean Decrease in Gini-Coefficient (using all variables)



2nd-Most Important Variable (mtry=2)
w.r.t. Mean Decrease in Gini-Coefficient (using only numerical variables)

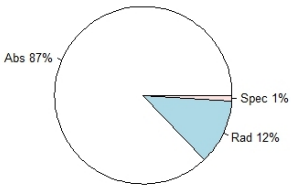


Number of variables considered while splitting a node is 2

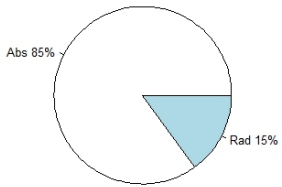
Random Forest(s)

Most Important Variable (as per voting)

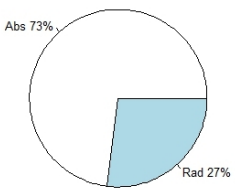
Most Important Variable (mtry=3)
w.r.t. Mean Decrease in Accuracy (using all variables)



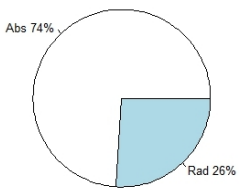
Most Important Variable (mtry=3)
w.r.t. Mean Decrease in Accuracy (using only numerical variables)



Most Important Variable (mtry=3)
w.r.t. Mean Decrease in Gini-Coefficient (using all variables)

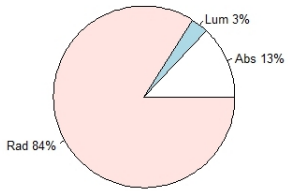


Most Important Variable (mtry=3)
w.r.t. Mean Decrease in Gini-Coefficient (using only numerical variable

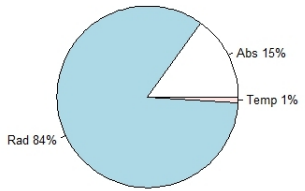


2nd-Most Important Variable (as per voting)

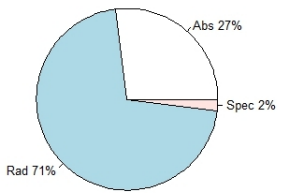
2nd-Most Important Variable (mtry=3)
w.r.t. Mean Decrease in Accuracy (using all variables)



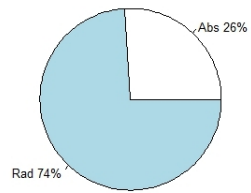
2nd-Most Important Variable (mtry=3)
w.r.t. Mean Decrease in Accuracy (using only numerical variables)



2nd-Most Important Variable (mtry=3)
w.r.t. Mean Decrease in Gini-Coefficient (using all variables)



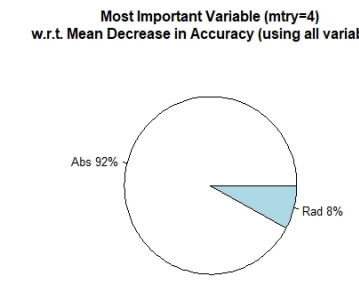
2nd-Most Important Variable (mtry=3)
w.r.t. Mean Decrease in Gini-Coefficient (using only numerical variables)

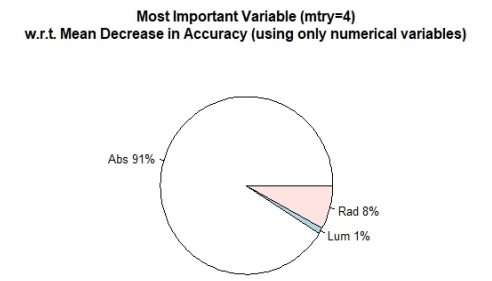


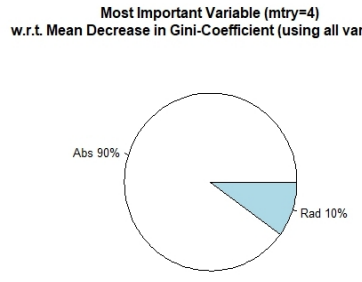
Number of variables considered while splitting a node is 3

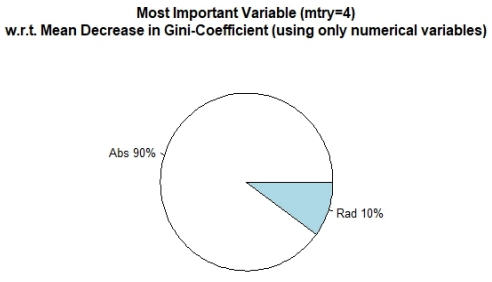
Random Forest(s)

Most Important Variable (as per voting)

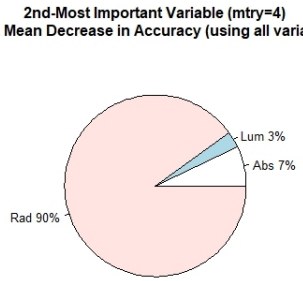


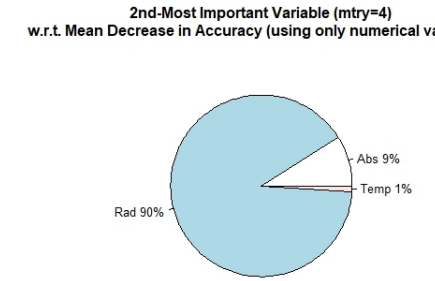


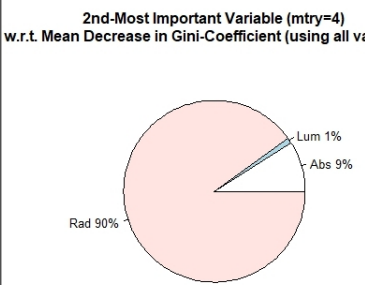


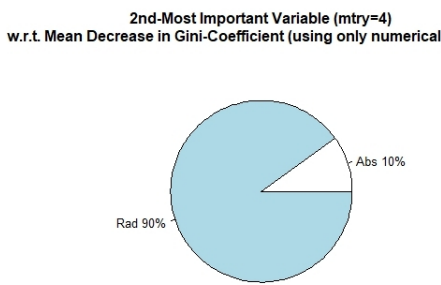


2nd-Most Important Variable (as per voting)









Number of variables considered while splitting a node is 4

Artificial Neural Network(s)

Data Pre-Processing necessary for ANN

```
> sort(tapply(data.play$Temp,data.play$Col,median))
                red              orange           yellowish          orange red
             3324.0              4018.0              4526.0              5112.0
      yellow white       white yellow pale yellow orange             whitish
             6158.0              7100.0              7230.0              8876.0
              white    yellowish white         blue white                blue
             8879.5             11790.0             14100.0             19921.5
> sort(tapply(data.play$Temp,data.play$Spec,median))
      M       K       G       F       A       B       O
 3324.0  4406.5  6850.0  7230.0  9030.0 18850.0 22369.0
```

"Star Color" and "Star Spectral Class" are actually ordinal categorical variables, theoretically, and there is also empirical evidence in favour of that

```
> library(Dict)
  dictionary.Spec=dict("M"=1,"K"=2,"G"=3,"F"=4,"A"=5,"B"=6,"O"=7)
  dictionary.Col = dict("red"=1,"orange"=2,"yellowish"=3,"orange red"=4,"yellow white"=5,"white
  yellow"=6,"pale yellow orange"=7,"whitish"=8,"white"=9,"yellowish white"=10,"blue white"=11
  ,"blue"=12)
  for(i in 1:dim(data.train)[1]){
      data.train$Col[i]=dictionary.Col[data.train$Col[i]]
      data.train$Spec[i]=dictionary.Spec[data.train$Spec[i]]
  }
  for(i in 1:dim(data.test)[1]){
      data.test$Col[i]=dictionary.Col[data.test$Col[i]]
      data.test$Spec[i]=dictionary.Spec[data.test$Spec[i]]
  }
  data.train$Col=as.numeric(data.train$Col)
  data.train$Spec=as.numeric(data.train$Spec)
  data.test$Col=as.numeric(data.test$Col)
  data.test$Spec=as.numeric(data.test$Spec)
```

Taking advantage of their ordinal nature, both variables have their labels changed with appropriate numerical values as per the order relation

```
> table(rbind(data.train,data.test)$Col)

  1   2   3   4   5   6   7   8   9  10  11  12
112   2   3   1   8   1   1   2  10   3  41  56
> table(rbind(data.train,data.test)$Spec)

  1   2   3   4   5   6   7
111   6   1  17  19  46  40
```

The training data and the testing data, both have their above-mentioned-categorical-variables appropriately modified

R Package(s) used:
->Dict: to implement the 'dictionary' data-type

Overview of ANN architecture

```
#build model-structure
model <- keras_model_sequential()
model %>%
    layer_dense(units = 20, activation = "sigmoid", input_shape = 6) %>%
    layer_dropout(rate = 0.3) %>%
    layer_dense(units = 20, activation = "sigmoid") %>%
    layer_dropout(rate = 0.3) %>%
    layer_dense(units = 6,activation = "sigmoid")
# view model layers
summary(model)
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_2 (Dense)              (None, 20)                140
dropout_1 (Dropout)          (None, 20)                0
dense_1 (Dense)              (None, 20)                420
dropout (Dropout)            (None, 20)                0
dense (Dense)                (None, 6)                 126
=================================================================
Total params: 686
Trainable params: 686
Non-trainable params: 0
```

```
# compile model
model %>% compile(
    loss = "sparse_categorical_crossentropy",
    optimizer = "adam",
    metrics = c("accuracy")
)
```

As the response-labels are integers and not one-hot-encoded, *sparse categorical crossentropy* is the appropriate loss function instead of *categorical crossentropy*

```
# fit model
model %>% fit(
    x = as.matrix(data.train[,-c(7)],nrow=dim(data.train)[1],ncol=dim(data.train)[2]-1,byrow = T),
    y = data.train$Type,
    epochs = 25*10, #number of times the learning algorithm will work through the entire dataset
    verbose = 1 #decides how to show the ongoing performance
)
```

Input always has to be an appropriate-sized numerical matrix (non-numerical gives error)

R Package(s) used:
->keras and tensorflow: integrated with Python to provide smooth performance

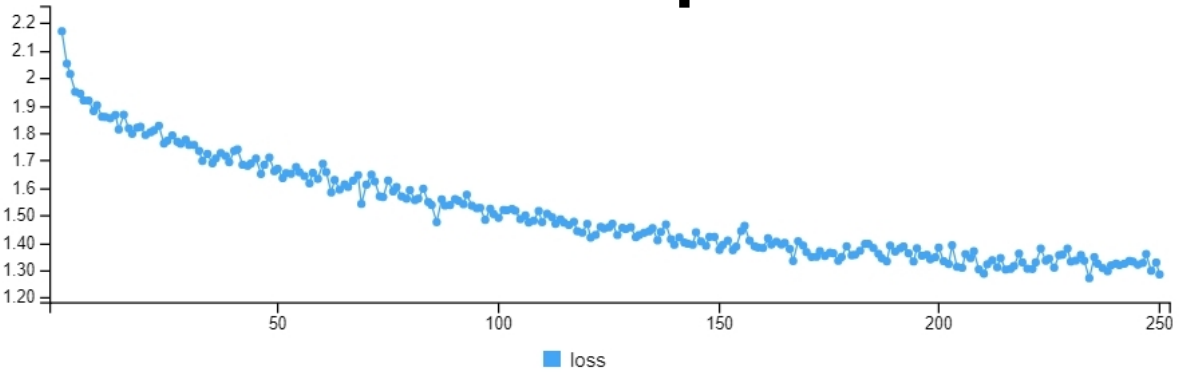
ANN with raw data as input

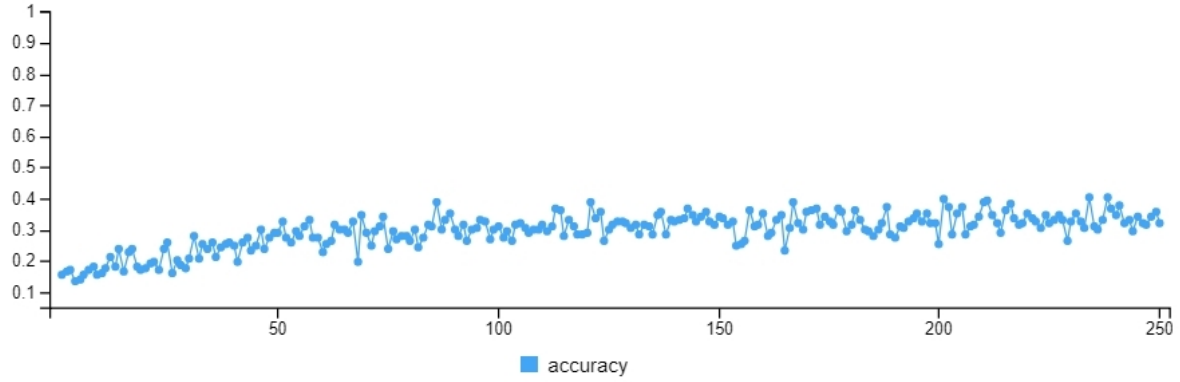
->Poor performance

->Normalizing the data is helpful

->Increasing the number of *epochs* is helpful

->Changing activation-functions is helpful





Here both the *x-axes* denote the the number of times the learning algorithm has run through the dataset

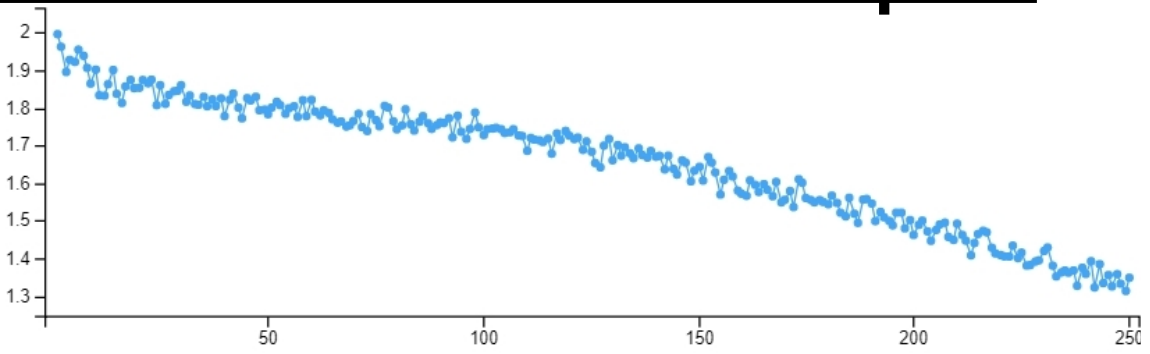
```
> error.train
     loss  accuracy
1.2241926 0.3626943
> error.test
     loss  accuracy
1.2543011 0.3191489
> cmatrix
                  Red Dwarf Brown Dwarf White Dwarf Main Sequence SuperGiants HyperGiants
Red Dwarf(P)              8           8           9             6           0           0
Brown Dwarf(P)            0           0           0             0           0           0
White Dwarf(P)            0           0           0             0           0           0
Main Sequence(P)          0           0           0             1           0           0
SuperGiants(P)            0           0           0             0           0           0
HyperGiants(P)            0           0           0             1           8           6
```

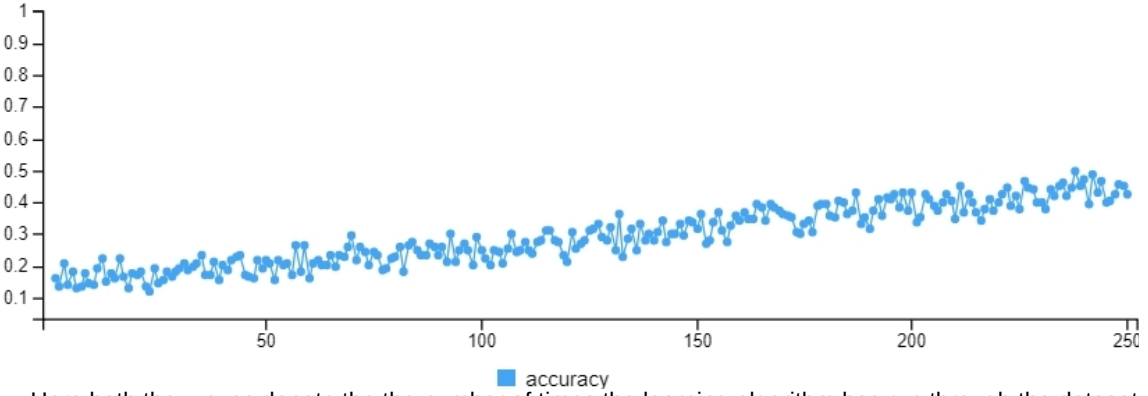
R Package(s) used:
->keras and tensorflow: integrated with Python to provide smooth performance

ANN with normalized data as input

```
max.Stuff=
  as.numeric(apply(rbind(data.train[,-7],data.test[,-7]),2,max))
min.Stuff=
  as.numeric(apply(rbind(data.train[,-7],data.test[,-7]),2,min))
for(i in 1:6){
  data.train[,i]=
    (data.train[,i]-min.Stuff[i])/(max.Stuff[i]-min.Stuff[i])
  data.test[,i]=
    (data.test[,i]-min.Stuff[i])/(max.Stuff[i]-min.Stuff[i])
}
```

Each (independent) variable has its minimum value in dataset subtracted from it, then it is divided by its maximum value in the dataset





Here both the *x-axes* denote the the number of times the learning algorithm has run through the dataset

->Results are better
->Further increasing the *epochs* is helpful
->Playing with *architecture* is helpful

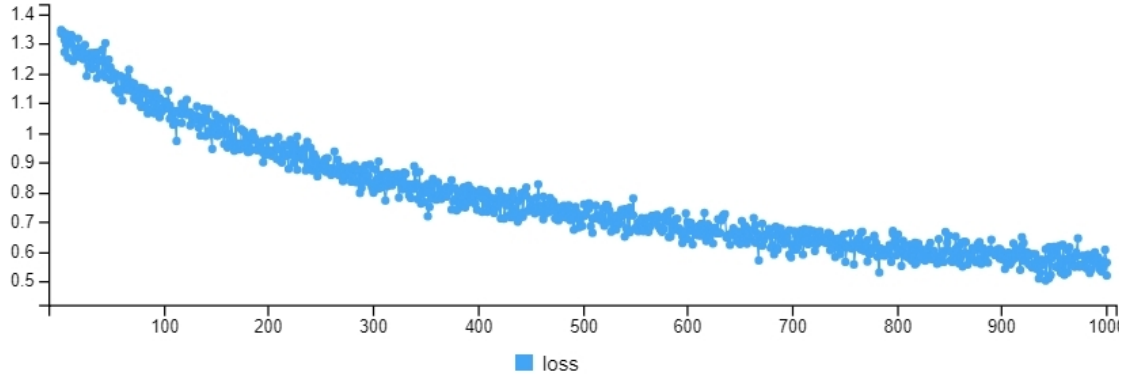
```
> error.train
     loss  accuracy
1.2018551 0.6165803
> error.test
     loss  accuracy
1.2410249 0.5744681
> cmatrix
                 Red Dwarf Brown Dwarf White Dwarf Main Sequence SuperGiants HyperGiants
Red Dwarf(P)             8           8           0             1           0           0
Brown Dwarf(P)           0           0           0             0           0           0
White Dwarf(P)           0           0           9             5           0           0
Main Sequence(P)         0           0           0             0           0           0
SuperGiants(P)           0           0           0             2           6           2
HyperGiants(P)           0           0           0             0           2           4
```

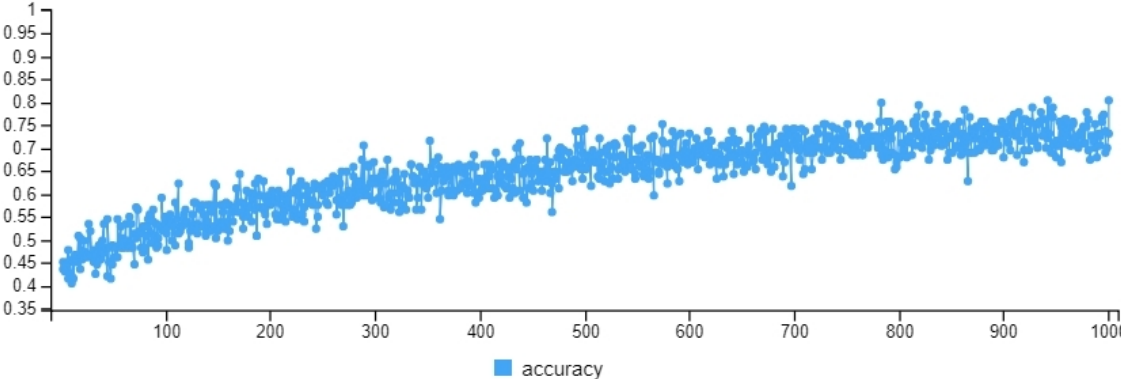
R Package(s) used:
->keras and tensorflow: integrated with Python to provide smooth performance

ANN with normalized data as input

```
max.Stuff=
  as.numeric(apply(rbind(data.train[,-7],data.test[,-7]),2,max))
min.Stuff=
  as.numeric(apply(rbind(data.train[,-7],data.test[,-7]),2,min))
for(i in 1:6){
  data.train[,i]=
    (data.train[,i]-min.Stuff[i])/(max.Stuff[i]-min.Stuff[i])
  data.test[,i]=
    (data.test[,i]-min.Stuff[i])/(max.Stuff[i]-min.Stuff[i])
}
```

Each (independent) variable has its minimum value in dataset subtracted from it, then it is divided by its maximum value in the dataset





Here both the *x-axes* denote the the number of times the learning algorithm has run through the dataset

->Results are better
->Further increasing the *epochs* may be helpful, but it is costly

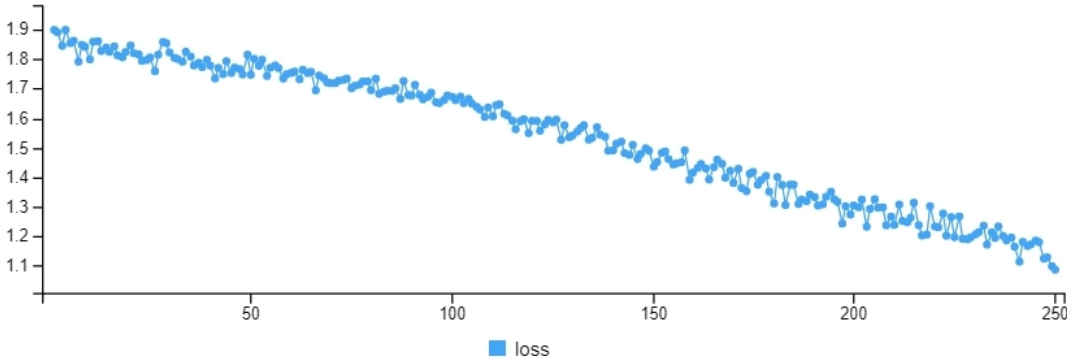
```
> error.train
      loss  accuracy
0.4012301 0.8186529
> error.test
      loss  accuracy
0.4139560 0.8297873
> cmatrix
                 Red Dwarf Brown Dwarf White Dwarf Main Sequence SuperGiants HyperGiants
Red Dwarf(P)             2           0           0             0           0           0
Brown Dwarf(P)           6           8           0             1           0           0
White Dwarf(P)           0           0           9             0           0           0
Main Sequence(P)         0           0           0             6           0           0
SuperGiants(P)           0           0           0             1           8           0
HyperGiants(P)           0           0           0             0           0           6
```

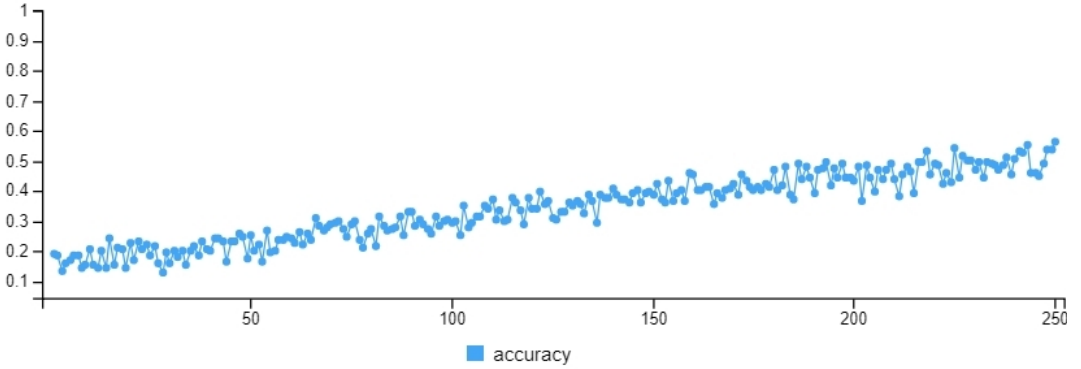
R Package(s) used:
->keras and tensorflow: integrated with Python to provide smooth performance

ANN with normalized data as input

```
max.Stuff=as.numeric(apply(rbind(data.train[,-7],data.test[,-7]),2,max))
for(i in 1:6){
  data.train[,i]=data.train[,i]/(max.Stuff[i])
  data.test[,i]=data.test[,i]/(max.Stuff[i])
}
```

Each (independent) variable is divided by its maximum value in the dataset





->Results are better than earlier normalization
->Further increasing the *epochs* is helpful
->Playing with *architecture* is helpful

```
> error.train
     loss accuracy
1.010610 0.626943
> error.test
      loss  accuracy
1.0496489 0.6170213
> cmatrix
                 Red Dwarf Brown Dwarf White Dwarf Main Sequence SuperGiants HyperGiants
Red Dwarf(P)             5           8           0             0           0           0
Brown Dwarf(P)           3           0           0             0           0           0
White Dwarf(P)           0           0           9             3           0           0
Main Sequence(P)         0           0           0             3           0           0
SuperGiants(P)           0           0           0             2           6           0
HyperGiants(P)           0           0           0             0           2           6
```

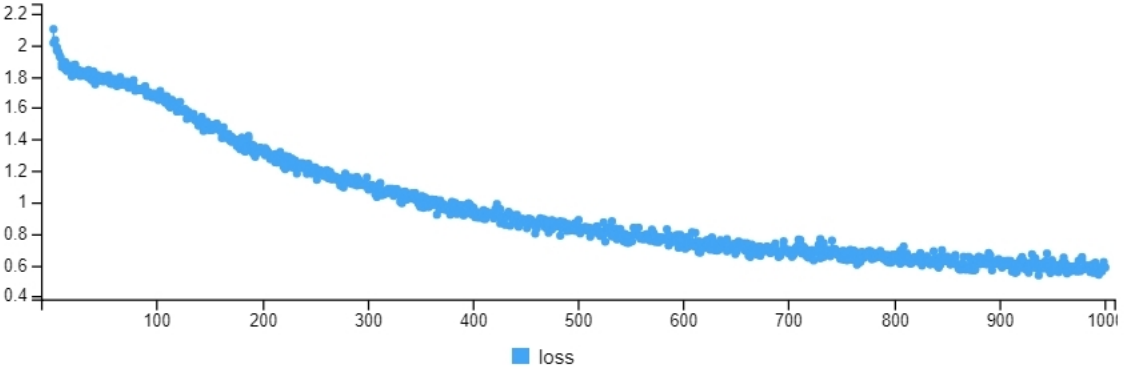
Here both the *x-axes* denote the the number of times the learning algorithm has run through the dataset

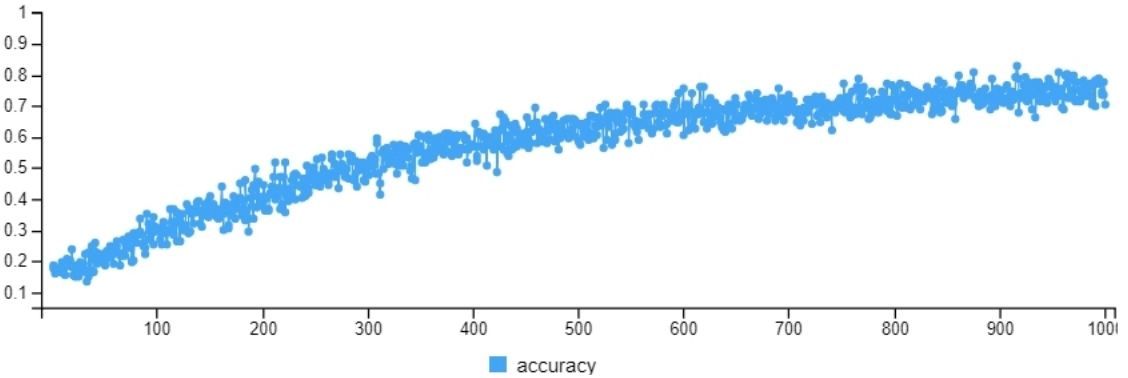
R Package(s) used:
->keras and tensorflow: integrated with Python to provide smooth performance

ANN with normalized data as input

```
max.Stuff=as.numeric(apply(rbind(data.train[,-7],data.test[,-7]),2,max))
for(i in 1:6){
  data.train[,i]=data.train[,i]/(max.Stuff[i])
  data.test[,i]=data.test[,i]/(max.Stuff[i])
}
```

Each (independent) variable is divided by its maximum value in the dataset





Here both the *x-axes* denote the the number of times the learning algorithm has run through the dataset

->Results are better than earlier normalization
->Further increasing the *epochs* may be helpful, but it will be costly

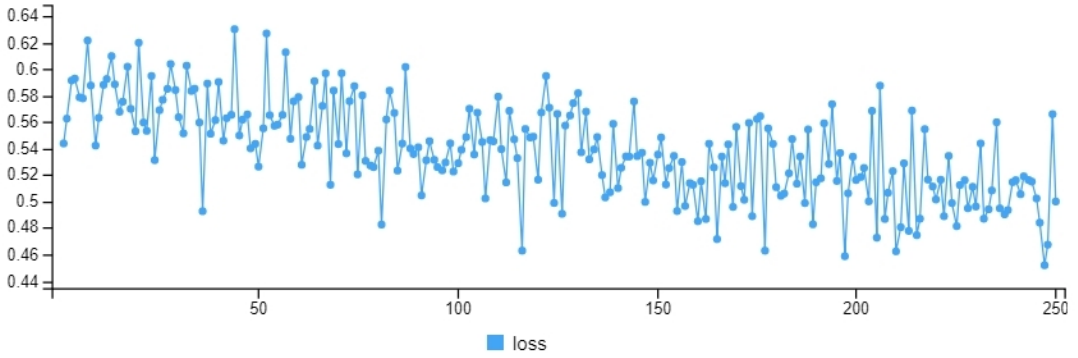
```
> error.train
     loss  accuracy
0.4014740 0.9015544
> error.test
     loss  accuracy
0.4063254 0.9148936
> cmatrix
                 Red Dwarf Brown Dwarf White Dwarf Main Sequence SuperGiants HyperGiants
Red Dwarf(P)             8           4           0             0           0           0
Brown Dwarf(P)           0           4           0             0           0           0
White Dwarf(P)           0           0           9             0           0           0
Main Sequence(P)         0           0           0             8           0           0
SuperGiants(P)           0           0           0             0           8           0
HyperGiants(P)           0           0           0             0           0           6
```

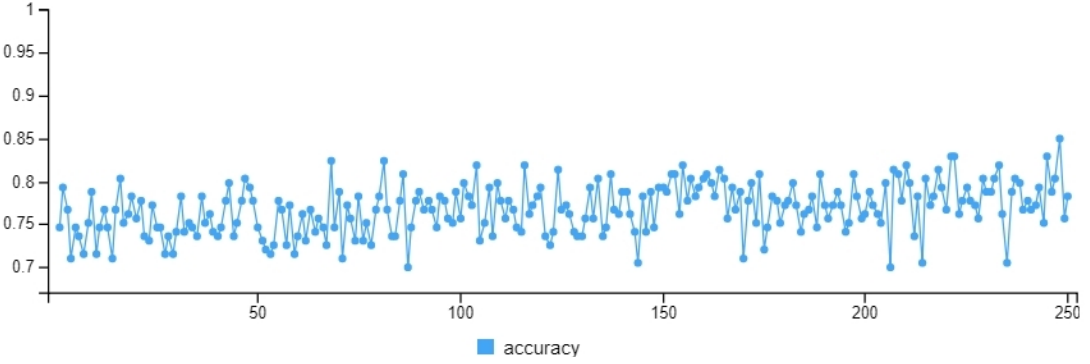
R Package(s) used:
->keras and tensorflow: integrated with Python to provide smooth performance

ANN with normalized data as input

```
max.Stuff=as.numeric(apply(rbind(data.train[,-7],data.test[,-7]),2,max))
for(i in 1:6){
  data.train[,i]=data.train[,i]/(max.Stuff[i])
  data.test[,i]=data.test[,i]/(max.Stuff[i])
}
```

Each (independent) variable is divided by its maximum value in the dataset





Here both the *x-axes* denote the the number of times the learning algorithm has run through the dataset

->Results are better than earlier normalization
->No change in accuracy even if epochs are increased beyond this

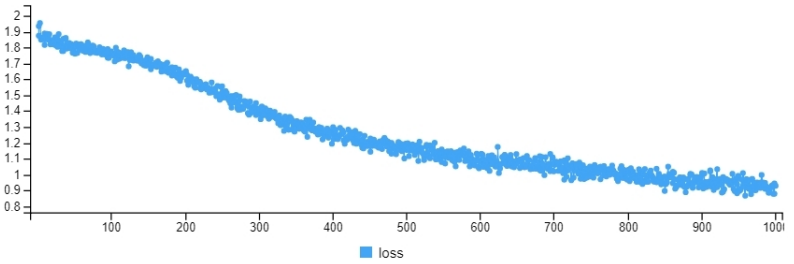
```
> error.train
      loss  accuracy
0.2558638 0.9637306
> error.test
      loss  accuracy
0.2564179 0.9574468
> cmatrix
                 Red Dwarf Brown Dwarf White Dwarf Main Sequence SuperGiants HyperGiants
Red Dwarf(P)             8           2           0             0           0           0
Brown Dwarf(P)           0           6           0             0           0           0
White Dwarf(P)           0           0           9             0           0           0
Main Sequence(P)         0           0           0             8           0           0
SuperGiants(P)           0           0           0             0           8           0
HyperGiants(P)           0           0           0             0           0           6
```

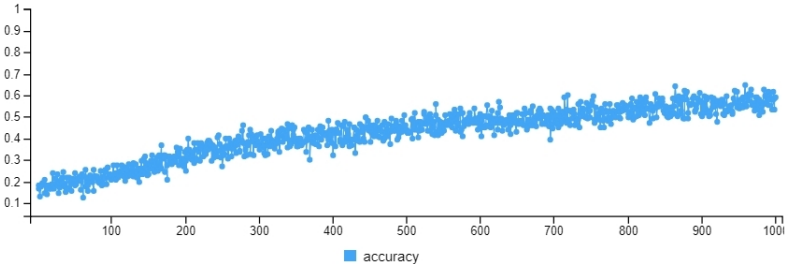
R Package(s) used:
->keras and tensorflow: integrated with Python to provide smooth performance

ANN with changed architecture

```
max.Stuff=as.numeric(apply(rbind(data.train[,-7],data.test[,-7]),2,max))
for(i in 1:6){
  data.train[,i]=data.train[,i]/(max.Stuff[i])
  data.test[,i]=data.test[,i]/(max.Stuff[i])
}
```

Each (independent) variable is divided by its maximum value in the dataset





Here both the *x-axes* denote the the number of times the learning algorithm has run through the dataset

```
> summary(model)
Model: "sequential_2"
_________________________________________________________________________________
 Layer (type)                        Output Shape                      Param #
=================================================================================
 dense_8 (Dense)                     (None, 10)                        70
 dropout_5 (Dropout)                 (None, 10)                        0
 dense_7 (Dense)                     (None, 10)                        110
 dropout_4 (Dropout)                 (None, 10)                        0
 dense_6 (Dense)                     (None, 6)                         66
=================================================================================
Total params: 246
Trainable params: 246
Non-trainable params: 0
_________________________________________________________________________________
> error.train
     loss  accuracy
0.6796284 0.7668394
> error.test
     loss  accuracy
0.6803305 0.7659575
> cmatrix
                Red Dwarf Brown Dwarf White Dwarf Main Sequence SuperGiants HyperGiants
Red Dwarf(P)            8           8           0             0           0           0
Brown Dwarf(P)          0           0           0             0           0           0
White Dwarf(P)          0           0           9             1           0           0
Main Sequence(P)        0           0           0             7           0           0
SuperGiants(P)          0           0           0             0           6           0
HyperGiants(P)          0           0           0             0           2           6
```

Not promising enough

R Package(s) used:
->keras and tensorflow: integrated with Python to provide smooth performance

ANN with changed architecture

```
max.Stuff=as.numeric(apply(rbind(data.train[,-7],data.test[,-7]),2,max))
for(i in 1:6){
  data.train[,i]=data.train[,i]/(max.Stuff[i])
  data.test[,i]=data.test[,i]/(max.Stuff[i])
}
```

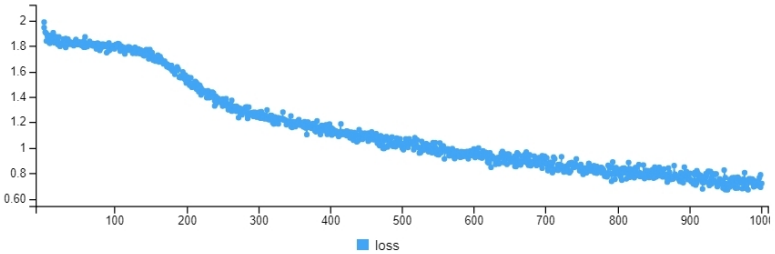
Each (independent) variable is divided by its maximum value in the dataset

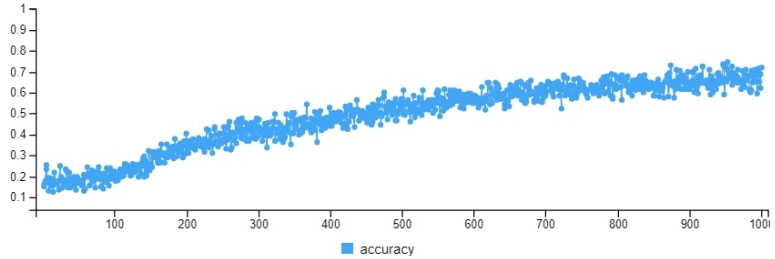
```
> summary(model)
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 dense_12 (Dense)             (None, 20)                140
 dropout_8 (Dropout)          (None, 20)                0
 dense_11 (Dense)             (None, 20)                420
 dropout_7 (Dropout)          (None, 20)                0
 dense_10 (Dense)             (None, 20)                420
 dropout_6 (Dropout)          (None, 20)                0
 dense_9 (Dense)              (None, 6)                 126
=================================================================
Total params: 1,106
Trainable params: 1,106
Non-trainable params: 0
_________________________________________________________________
```

```
> error.train
     loss  accuracy
0.4815640 0.8445596
> error.test
     loss  accuracy
0.5010631 0.8297873
```

Not promising enough

```
> cmatrix
                 Red Dwarf Brown Dwarf White Dwarf Main Sequence SuperGiants HyperGiants
Red Dwarf(P)             8           7           0             0           0           0
Brown Dwarf(P)           0           1           0             0           0           0
White Dwarf(P)           0           0           9             1           0           0
Main Sequence(P)         0           0           0             7           0           0
SuperGiants(P)           0           0           0             0           8           0
HyperGiants(P)           0           0           0             0           0           6
```





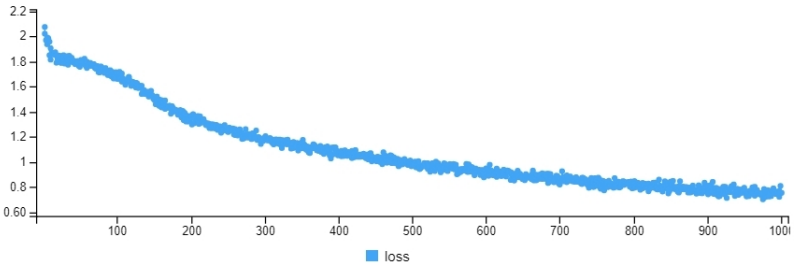
Here both the *x-axes* denote the the number of times the learning algorithm has run through the dataset

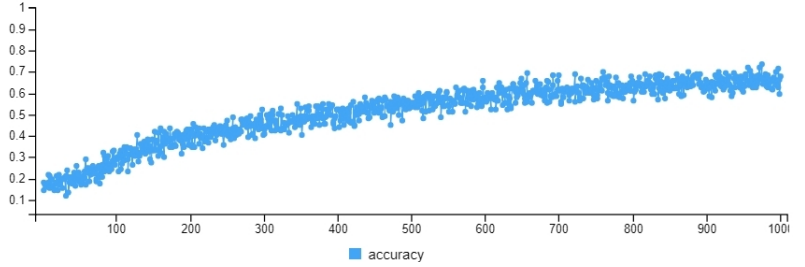
R Package(s) used:
->keras and tensorflow: integrated with Python to provide smooth performance

ANN with only 4 numerical variables

```
max.Stuff=as.numeric(apply(rbind(data.train[,-7],data.test[,-7]),2,max))
for(i in 1:6){
  data.train[,i]=data.train[,i]/(max.Stuff[i])
  data.test[,i]=data.test[,i]/(max.Stuff[i])
}
```

Each (independent) variable is divided by its maximum value in the dataset





Here both the *x-axes* denote the the number of times the learning algorithm has run through the dataset

```
> summary(model)
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 dense_15 (Dense)             (None, 20)                100
 dropout_10 (Dropout)         (None, 20)                0
 dense_14 (Dense)             (None, 20)                420
 dropout_9 (Dropout)          (None, 20)                0
 dense_13 (Dense)             (None, 6)                 126
=================================================================
Total params: 646
Trainable params: 646
Non-trainable params: 0
_________________________________________________________________
> error.train
     loss  accuracy
0.5894013 0.9119171
> error.test
     loss  accuracy
0.6215589 0.8936170
> cmatrix
                 Red Dwarf Brown Dwarf White Dwarf Main Sequence SuperGiants HyperGiants
Red Dwarf(P)             8           1           0             0           0           0
Brown Dwarf(P)           0           7           4             0           0           0
White Dwarf(P)           0           0           5             0           0           0
Main Sequence(P)         0           0           0             8           0           0
SuperGiants(P)           0           0           0             0           8           0
HyperGiants(P)           0           0           0             0           0           6
```

Promising enough to run some more *epochs*

R Package(s) used:
->keras and tensorflow: integrated with Python to provide smooth performance

ANN with only 4 numerical variables

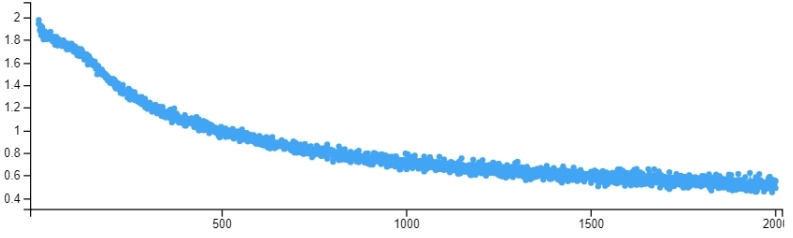
```
max.Stuff=as.numeric(apply(rbind(data.train[,-7],data.test[,-7]),2,max))
for(i in 1:6){
  data.train[,i]=data.train[,i]/(max.Stuff[i])
  data.test[,i]=data.test[,i]/(max.Stuff[i])
}
```

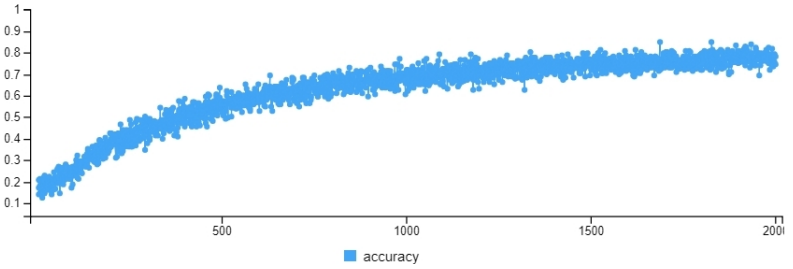
Each (independent) variable is divided by its maximum value in the dataset

->Full data performed slightly better

->Some more insights into the data might have led in choice of better activation functions

->Using slightly adulterated observations to increase training sample size might have helped





Here both the *x-axes* denote the the number of times the learning algorithm has run through the dataset

```
> error.train
     loss  accuracy
0.3218408 0.9222798
> error.test
     loss  accuracy
0.3248554 0.9148936
> cmatrix
                 Red Dwarf Brown Dwarf White Dwarf Main Sequence SuperGiants HyperGiants
Red Dwarf(P)             8           2           0             0           0           0
Brown Dwarf(P)           0           6           2             0           0           0
White Dwarf(P)           0           0           7             0           0           0
Main Sequence(P)         0           0           0             8           0           0
SuperGiants(P)           0           0           0             0           8           0
HyperGiants(P)           0           0           0             0           0           6
```

R Package(s) used:
->keras and tensorflow: integrated with Python to provide smooth performance

Fitting LDA

The LDA result on training data is :

```
Call:
lda(Y ~ ., data = Train.Data)

Prior probabilities of groups:
        0         1         2         3         4         5
0.1614583 0.1979167 0.1614583 0.1354167 0.1718750 0.1718750

Group means:
  Temperature..K. Luminosity.L.Lo. Radius.R.Ro. Absolute.magnitude.Mv.
0        2995.226     7.168065e-04 1.075742e-01             17.3561290
1        3304.000     5.648158e-03 3.546789e-01             12.5533947
2       13905.516     7.690323e-04 1.087032e-02             12.6106452
3       14818.769     1.865543e+04 4.307077e+00             -0.3759231
4       16101.758     3.088479e+05 4.912121e+01             -6.3841515
5       12030.636     3.139828e+05 1.369133e+03             -9.6090909

Coefficients of linear discriminants:
                                  LD1           LD2           LD3          LD4
Temperature..K.        -4.047161e-05 -3.015465e-06  1.007611e-04 9.625885e-05
Luminosity.L.Lo.        1.916904e-06 -9.128886e-07 -8.298416e-06 3.665357e-06
Radius.R.Ro.            4.565648e-03  8.117468e-03  8.639493e-04 3.342672e-04
Absolute.magnitude.Mv. -5.135532e-01  3.287780e-01 -4.167660e-02 9.180538e-02

Proportion of trace:
   LD1    LD2    LD3    LD4
0.8185 0.1722 0.0073 0.0020
```

The prior probabilities along with the group means and linear discriminants are shown in the above image.

Error Rate of LDA

The confusion matrix below is for Training set.The misclassication rate here is 0.02083333.

Train.Y \ pred	0	1	2	3	4	5
0	31	0	0	0	0	0
1	0	38	0	0	0	0
2	0	4	27	0	0	0
3	0	0	0	26	0	0
4	0	0	0	0	33	0
5	0	0	0	0	0	33

The confusion matrix below is for Test set.The misclassication rate here is 0.04166667.

Test.Y \ p	0	1	2	3	4	5
0	9	0	0	0	0	0
1	0	2	0	0	0	0
2	0	0	9	0	0	0
3	0	2	0	12	0	0
4	0	0	0	0	7	0
5	0	0	0	0	0	7

Visualizing LDA

Since we have 4 predictors here,it is not possible to visualize the classification by drawing it in 2D or 3D.But we can do it using PCA.We plan to convert the given data into a 2D data using the first two principal components and then show our LDA classifications on Test Data.

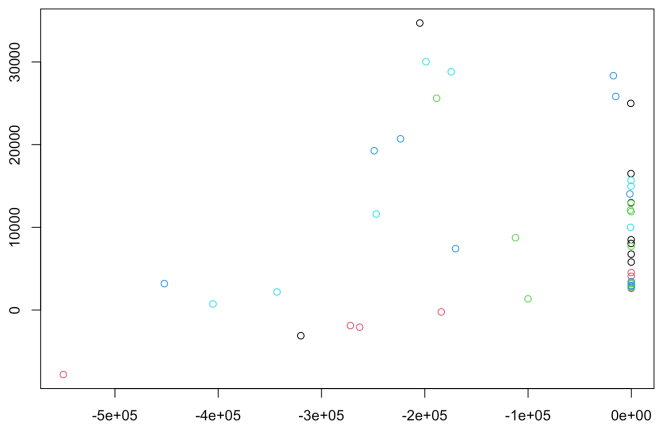


Figure: Given Data

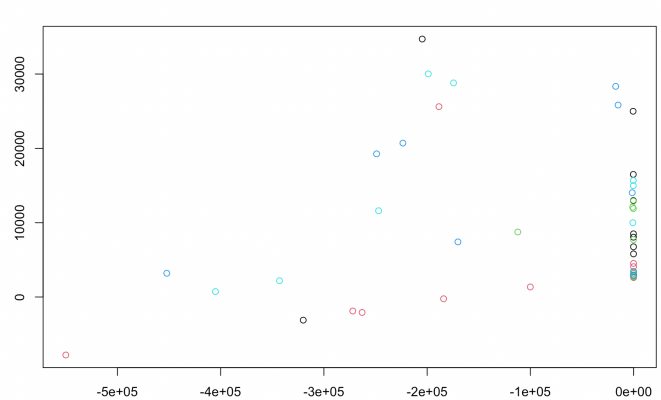


Figure: Classification

Fitting QDA

The QDA result on training data is :

```
Call:
qda(Y ~ ., data = Train.Data)

Prior probabilities of groups:
        0         1         2         3         4         5
0.1510417 0.1770833 0.1562500 0.1770833 0.1770833 0.1614583

Group means:
  Temperature..K. Luminosity.L.Lo. Radius.R.Ro. Absolute.magnitude.Mv.
0        2971.379     6.895517e-04 1.094276e-01              17.7027586
1        3273.882     5.093529e-03 3.504353e-01              12.4436176
2       14196.000     3.015167e-03 1.078967e-02              12.5040000
3       16698.853     3.729912e+04 4.364235e+00              -0.4505882
4       15461.029     3.156385e+05 4.850000e+01              -6.3784412
5       11729.000     3.163723e+05 1.366400e+03              -9.6935484
```

The prior probabilities along with the group means are shown in the above image.

Error Rate of QDA

The confusion matrix below is for Training set.The misclassication rate here is 0.005208333.

	Train.Y					
p.qda	0	1	2	3	4	5
0	28	0	0	0	0	0
1	1	34	0	0	0	0
2	0	0	30	0	0	0
3	0	0	0	34	0	0
4	0	0	0	0	34	0
5	0	0	0	0	0	31

The confusion matrix below is for Test set.The misclassication rate here is 0.02083333.

	Test.Y					
p.qda	0	1	2	3	4	5
0	10	0	0	0	0	0
1	1	6	0	0	0	0
2	0	0	10	0	0	0
3	0	0	0	6	0	0
4	0	0	0	0	6	0
5	0	0	0	0	0	9

Fitting FDA

One vs Rest

Here we calculated $\alpha = S^{-1}(\bar{x}_1 - \bar{x}_2)$ and β by minimising the Kolmogorov distance between the two empirical distributions.
The error rates of different FDA classifications on test data is given below:

Type	Error Rate
0 v Rest	0
1 v Rest	0.1875
2 v Rest	0.14583
3 v Rest	0.02083
4 v Rest	0.0416
5 v Rest	0.0416

k-NN classifier

k=1

```
> model.knn = knn3(Y~., data=Train.Data, k = 1)
> p = predict(model.knn, Train.Data[,-1], type = "class")
> (T = table(Train.Y,p))
       p
Train.Y  0  1  2  3  4  5
      0 29  0  0  0  0  0
      1  0 31  0  0  0  0
      2  0  0 35  0  0  0
      3  0  0  0 30  0  0
      4  0  0  0  0 32  0
      5  0  0  0  0  0 35
> 1-sum(diag(T))/sum(T)
[1] 0
> p = predict(model.knn, Test.Data[,-1], type = "class")
> (T = table(Test.Y,p))
      p
Test.Y 0 1 2 3 4 5
     0 5 6 0 0 0 0
     1 2 7 0 0 0 0
     2 0 0 5 0 0 0
     3 0 0 5 5 0 0
     4 0 0 0 0 6 2
     5 0 0 0 0 2 3
> 1-sum(diag(T))/sum(T)
[1] 0.3541667
```

From the confusion matrix, we get that the misclassification rate for k=1 is 0 for the training set and 0.3541667 for the test set.

k-NN classifier

k=3

```
> model.knn = knn3(Y~., data=Train.Data, k = 3)
> p = predict(model.knn, Train.Data[,-1], type = "class")
> (T = table(Train.Y,p))
       p
Train.Y  0  1  2  3  4  5
      0 26  3  0  0  0  0
      1  6 25  0  0  0  0
      2  0  0 34  1  0  0
      3  0  1  3 26  0  0
      4  0  0  0  0 22 10
      5  0  0  0  0  9 26
> 1-sum(diag(T))/sum(T)
[1] 0.171875
> p = predict(model.knn, Test.Data[,-1], type = "class")
> (T = table(Test.Y,p))
       p
Test.Y 0 1 2 3 4 5
     0 2 9 0 0 0 0
     1 1 8 0 0 0 0
     2 0 0 5 0 0 0
     3 0 0 7 3 0 0
     4 0 0 0 0 7 1
     5 0 0 0 0 3 2
> 1-sum(diag(T))/sum(T)
[1] 0.4375
```

From the confusion matrix, we get that the misclassification rate for k=3 is 0.171875 for the training set and 0.4375 for the test set. [Error rate is more compared to the previous case.]

k-NN classifier

k=10

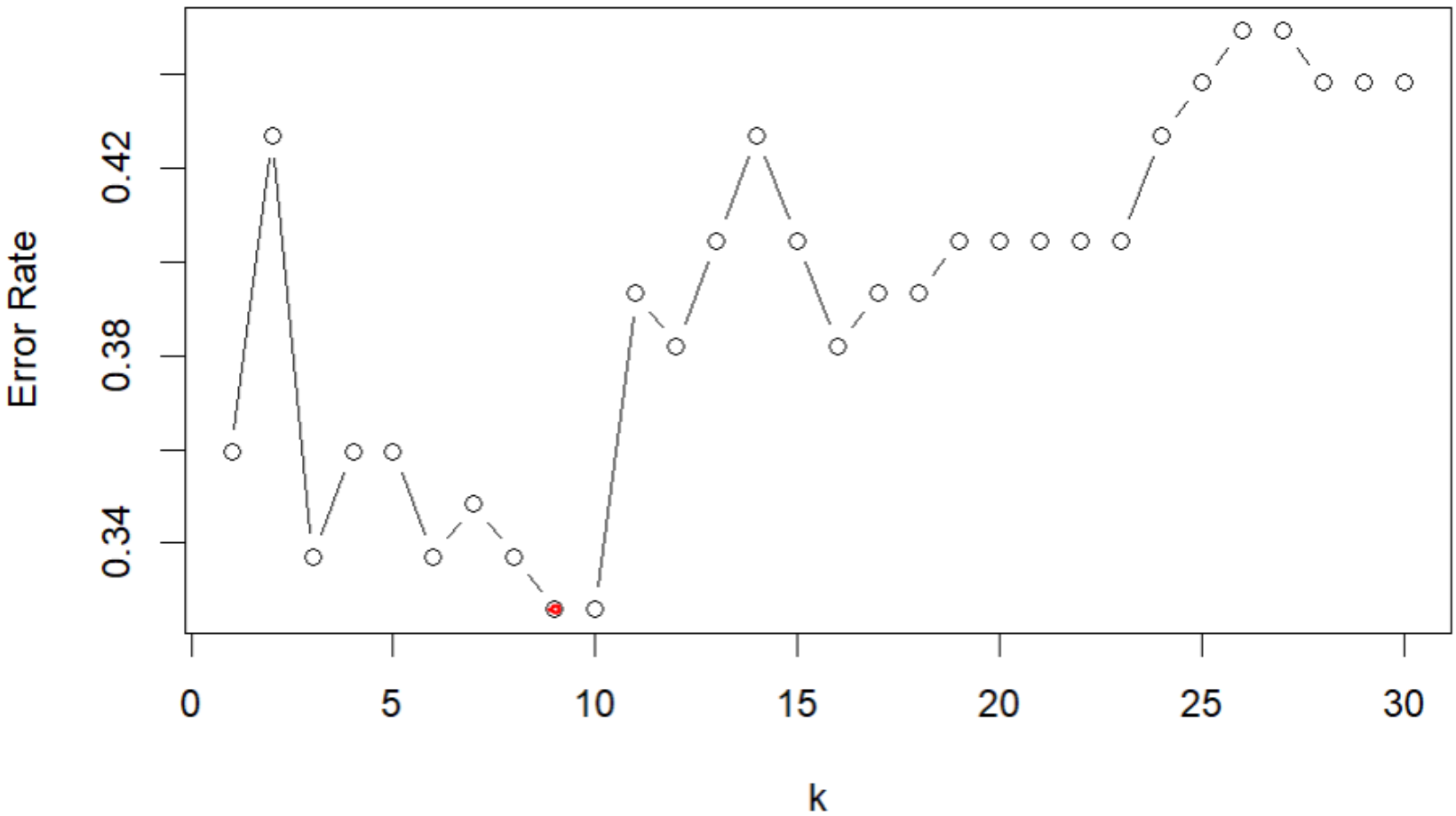
```
> model.knn = knn3(Y~., data=Train.Data, k = 10)
> p = predict(model.knn, Train.Data[,-1], type = "class")
> (T = table(Train.Y,p))
       p
Train.Y  0  1  2  3  4  5
      0 19 10  0  0  0  0
      1  7 24  0  0  0  0
      2  0  0 35  0  0  0
      3  0  1 11 18  0  0
      4  0  0  0  1 17 14
      5  0  0  0  0 15 20
> 1-sum(diag(T))/sum(T)
[1] 0.3072917
>
> p = predict(model.knn, Test.Data[,-1], type = "class")
> (T = table(Test.Y,p))
      p
Test.Y 0 1 2 3 4 5
     0 5 6 0 0 0 0
     1 1 8 0 0 0 0
     2 0 0 5 0 0 0
     3 0 1 6 3 0 0
     4 0 0 0 0 7 1
     5 0 0 0 0 2 3
> 1-sum(diag(T))/sum(T)
[1] 0.3541667
```

From the confusion matrix, we get that the misclassification rate for k=10 is 0.3072917 for the training set and 0.3541667 for the test set.

k-NN classifier

Optimal choice for k using cross-validation

The plot of error rate for different values of the number of nearest neighbors, k

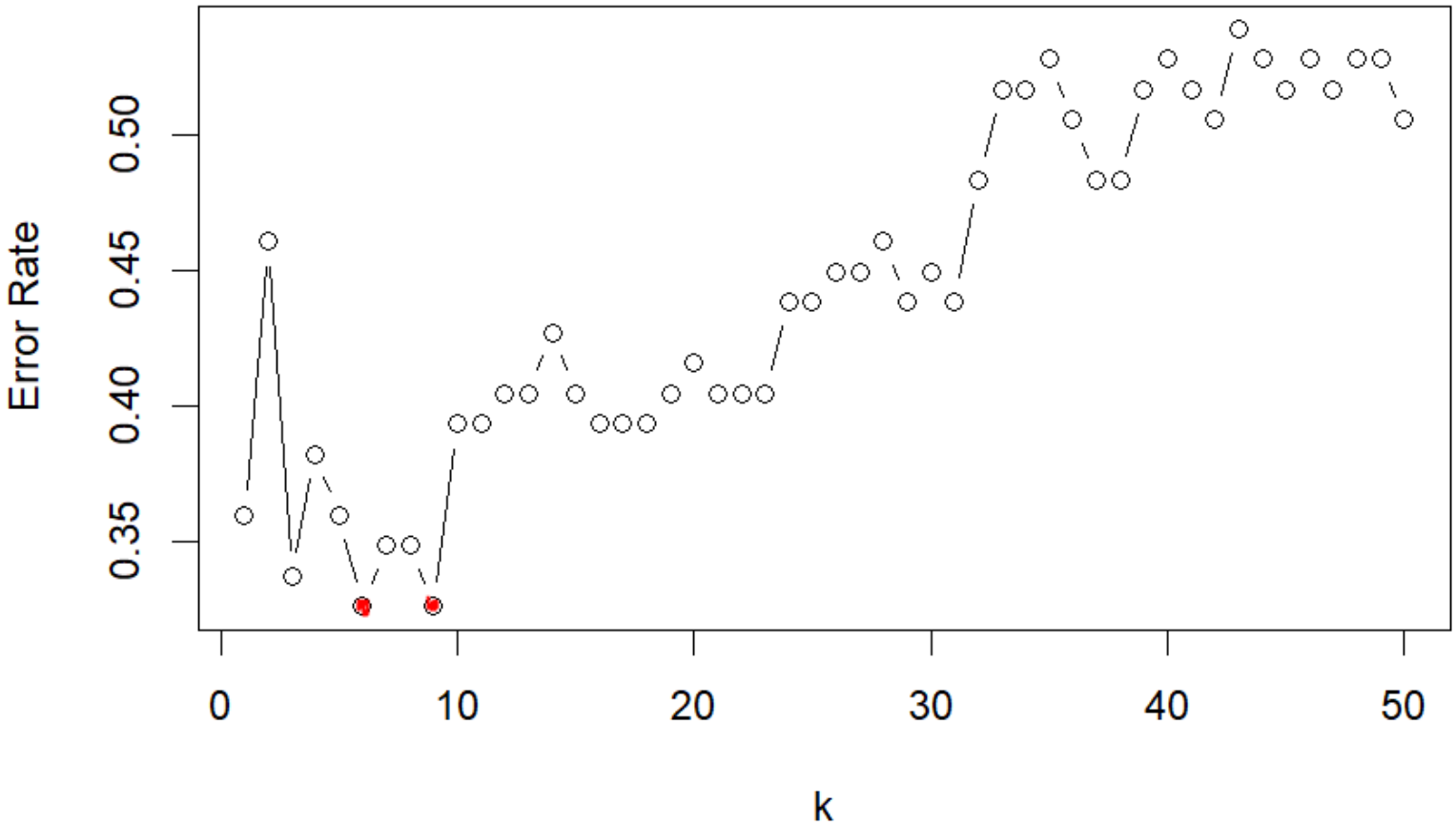


The error rate is minimum for k=9

k-NN classifier

Optimal choice for k using cross-validation

The plot of error rate for different values of the number of nearest neighbors k (upto k=50)



The error rate is minimum for k=6. But for k=9 the error rate is very close to that for k=6. There is also an advantage to choosing k as an odd number.

k-NN classifier

k=9

```
> model.knn = knn3(Y~., data=Train.Data, k = 9)
> p = predict(model.knn, Train.Data[,-1], type = "class")
> (T = table(Train.Y,p))
       p
Train.Y  0  1  4  5
      0 20 11  0  0
      1  9 26  0  0
      4  0  0  1  6
      5  0  0  0 16
> 1-sum(diag(T))/sum(T)
[1] 0.2921348
>
> p = predict(model.knn, Test.Data[,-1], type = "class")
> (T = table(Test.Y,p))
      p
Test.Y 0 1 4 5
     0 6 3 0 0
     1 1 4 0 0
     4 0 0 0 2
     5 0 1 0 6
> 1-sum(diag(T))/sum(T)
[1] 0.3043478
```

From the confusion matrix, we get that the misclassification rate for k=9 is 0.2921348 for the training set and 0.3043478 for the test set.

KDA

One vs Rest

- Chosen the Normal kernel
- Bandwidth matrix is obtained using the "Hkda" command in R which gives different bandwidths corresponding to different responses using least square cross-validation.
- Population proportions are used as the prior probabilities.

The error rates of different KDA classifications on test data are given below:

Type	Error Rate
0 v Rest	0.5
1 v Rest	0.5208333
2 v Rest	0.4375
3 v Rest	0.4791667
4 v Rest	0.2916667
5 v Rest	0.3125

KDA

One vs Rest

The misclassification probabilities are quite high in some cases. So KDA doesn't perform as good as the previous classifiers. One cause might be the overfitting.
The error rates of different KDA classifications on train data are given below:

Type	Error Rate
0 v Rest	0.046875
1 v Rest	0.1302083
2 v Rest	0
3 v Rest	0
4 v Rest	0
5 v Rest	0

Since, we have estimated priors as the population proportions, the classification based on Nadaraya-Watson estimate will be the same as KDA.

Fitting SVM with a Linear Kernel

The SVM result on Training data

```
Call:
svm(formula = Y ~ ., data = Train.Data, kernel = "linear", cost = 10^10)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  1e+10

Number of Support Vectors:  26

 ( 3 2 6 5 6 4 )


Number of Classes:  6

Levels:
 0 1 2 3 4 5
```

Error Rate of SVM (Linear Kernel)

The confusion matrix below is for Training set.The misclassication rate here is 0.

	Train.Y					
Prediction	0	1	2	3	4	5
0	34	0	0	0	0	0
1	0	34	0	0	0	0
2	0	0	33	0	0	0
3	0	0	0	32	0	0
4	0	0	0	0	31	0
5	0	0	0	0	0	28

The confusion matrix below is for Test set.The misclassication rate here is 0.

	Test.Y					
Prediction	0	1	2	3	4	5
0	6	0	0	0	0	0
1	0	6	0	0	0	0
2	0	0	7	0	0	0
3	0	0	0	8	0	0
4	0	0	0	0	9	0
5	0	0	0	0	0	12

Fitting SVM with a Radial Kernel

The SVM result on Training data

```
Call:
svm(formula = Y ~ ., data = Train.Data, kernel = "radial", cost = 10^10)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  radial
       cost:  1e+10

Number of Support Vectors:  44

 ( 3 11 12 5 9 4 )


Number of Classes:  6

Levels:
 0 1 2 3 4 5
```

Error Rate of SVM (Radial Kernel)

The confusion matrix below is for Training set.The misclassication rate here is 0.

	Train.Y					
Prediction	0	1	2	3	4	5
0	34	0	0	0	0	0
1	0	34	0	0	0	0
2	0	0	33	0	0	0
3	0	0	0	32	0	0
4	0	0	0	0	31	0
5	0	0	0	0	0	28

The confusion matrix below is for Test set.The misclassication rate here is 0.

	Test.Y					
Prediction	0	1	2	3	4	5
0	6	0	0	0	0	0
1	0	6	0	0	0	0
2	0	0	7	0	0	0
3	0	0	0	8	0	0
4	0	0	0	0	9	0
5	0	0	0	0	0	12

Comparing different Kernel functions

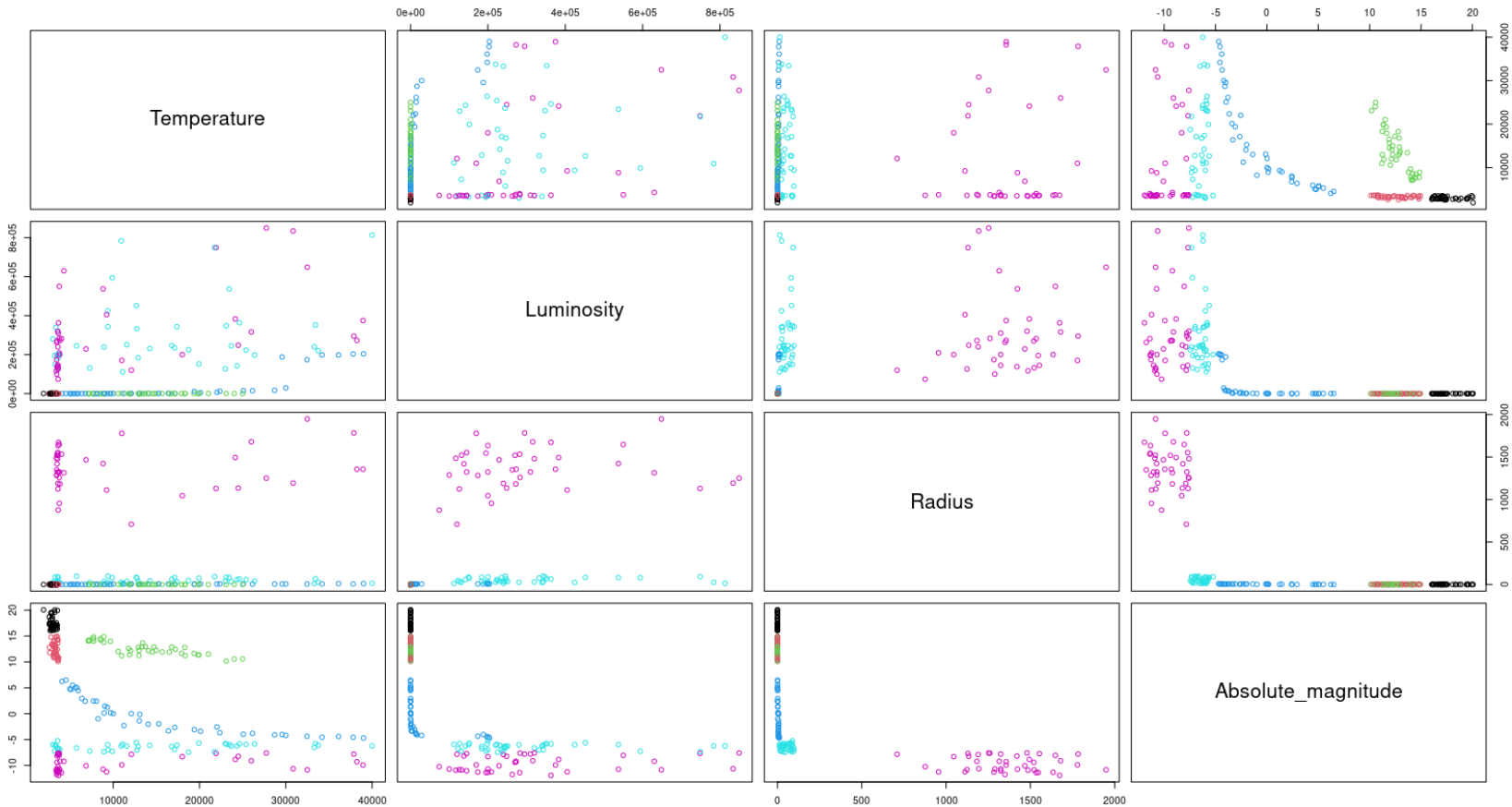
We compare 4 different kernel function using Cross-Validation to obtain following average misclassification rates.

Kernel Functions	Functional Formula	Average Misclassification Rate
Linear	$U'V$	0.004166667
Radial	$(\gamma U'V + c_0)^d$	0.006250000
Polynomial	$\exp\{ -\gamma \mid U - V \mid^2 \}$	0.012500000
Sigmoid	$\text{Tanh}\{ (\gamma U'V + c_0)^d \}$	0.081250000

SVM with a Linear Kernel Function has the least average misclassification rate amongst all.

Pairs Plot for the Numerical Variables

The figure shows pairwise scatter amongst the variables Temperature, Luminosity, Radius and the Absolute Magnitude.



Fitting Multinomial-Logistic Regerssion

Model Summary for the fitted multinomial-logistic model has been obtained as:

```
> summary(m)
Call:
multinom(formula = Y ~ ., data = Train.Data)

Coefficients:
  (Intercept)   Temperature    Luminosity      Radius Absolute_magnitude
1    235.9197  8.253768e-05 -0.0030081172 -0.28218567         -15.299435
2     18.3642  1.537377e-02 -0.0019589170 -0.01581338          -5.419474
3    238.0482  8.451101e-03 -0.0013114788 -1.25069573         -19.449424
4    288.3063 -1.733075e-03 -0.0002587371 -0.33286285         -26.192103
5    207.0996 -1.425312e-03 -0.0001450884 -0.12853728         -19.192598

Std. Errors:
   (Intercept) Temperature Luminosity     Radius Absolute_magnitude
1 5.975779e-01  0.08954076  0.1474303 1.40325134       1.713446e+01
2 1.221387e-03  0.08707845  0.7104603 0.00354937       2.235403e-02
3 2.692019e-01  0.04188525  0.6198668 0.70140171       1.039600e+01
4 9.708623e-04  5.97779359  0.4160565 0.05584059       7.299074e-04
5 3.564602e-05  7.18021694  0.4608076 0.03416086       7.342989e-04

Residual Deviance: 0.000799098
AIC: 50.0008
```

Error Rate of Logistic Regression Model

The confusion matrix below is for Training set.The misclassication rate here is 0.

	Train.Y					
Prediction	0	1	2	3	4	5
0	34	0	0	0	0	0
1	0	29	0	0	0	0
2	0	0	32	0	0	0
3	0	0	0	33	0	0
4	0	0	0	0	32	0
5	0	0	0	0	0	32

The confusion matrix below is for Test set.The misclassication rate here is 0.02083333.

	Test.Y					
Prediction	0	1	2	3	4	5
0	6	0	0	0	0	0
1	0	11	0	0	0	0
2	0	0	8	0	0	1
3	0	0	0	7	0	0
4	0	0	0	0	8	0
5	0	0	0	0	0	7

Average misclassification rate for Multinomial logistic model is 0.01777.

Acknowledgements

We like to thank **Professor Anil K. Ghosh** and **Bilol Banerjee** for giving us the opportunity to present this project and for their kind advice and support for this project.