Sorting Magic: The Gathering Cards by Color

Ted Tinker and Lilian Lu

December 2017

Abstract

Magic: The Gathering (MTG) is a collectible-card-game set in a fantasy multiverse. Since its introduction in 1993 more than 17,000 MTG cards have been designed and published in more than seventy sets. The cards vary in type and complexity, but all them have one of thirty-two Color Identities: either Colorless (C) or any combination of White (W) Blue (U) Black (B) Red (R) and Green (G). These colors relate to game-world idealogies, so the Color Identity of a card usually reflects what it might be able to do when played. A complete set of MTG card-data is available in a JSON file from mtgjson.com, but the JSON format is inconvenient in RStudio. In this project we parce the JSON file into a dataframe so we may use Rstudio to answer the following questions:

How does a card's Color Identity influence the rest of its statistics? Based on that, can a card's statistics be used to algorithmically predict its Color Identity? And, how do these questions relate to the design philosophy of Magic's creators? We elected to generate supervised classification models based on random forest and logistic regression. Once we select the better model, we'll examine its predictions to explain elements of Magic's design.

Our model consisting of six random forests (one to predict the presence of each of the five colors in a card's Color Identity, and the sixth to synthesize their outputs into a final guess) is quite proficient. On average its guess is only wrong by one color out of five.

Our model consisting of six logistic regressions has a slightly lower test error, but tends to guess single colors and therefore is not very accurate when applied to multicolored cards. Still, its predictions differ from test Color Identities by less than one color out of five.

Overall both models are surprisingly proficient, though the random forest model's predictions are difficult to justify while the logistic regression model tends to fail on multicolored cards (which make up a fair portion of the data).

Introduction

Our primary goal is to consistently algorithmically sort Magic: The Gathering cards by Color Identity based on the rest of their statistics (excluding statistics which make the problem nearly trivial). This is difficult because Magic: The Gathering cards come in a variety of types and levels of complexity, and Color Identity is the product of subjective game-design considerations by the designers. Expert humans are able to assign Color Identities to cards with ease; Mark Rosewater, Magic: The Gathering's head designer, arbitrates the color-philosophies with intricate precision. In "Mechanical Color Pie 2017" he associates every keyword in the game's standard lexicon with each color at one of four levels: primary, secondary, tertiary, or none of the above. Given the detailed knowledge required to accurately enforce the color-philosophies, an algorithm for assigning Color Identity based on card-data may be useful for designing the game or understanding the game's design.

This project is inspired by the RoboRosewater twitter account. This twitter account posts the output of a neural network trained on Magic: The Gathering cards. Some of RoboRosewater's cards are humorously unreadable; others are syntactically correct and theoretically playable. However, even RoboRosewater's best cards have mismatched Color Identities because the program cannot grasp the subtle intricacies of "flavor,"

or relation to in-game lore and philosophies. This project endeavors to improve automatic Color Identity identification.

The dataset of all Magic: The Gathering cards was downloaded from mtgjson.com in the form of a JSON file. JSON is a natural choice for Magic: The Gathering cards because the cards have inconsistant variables based on their card-type. Some variables have high consistency (for example, each card has a name), but some variables may not be present on all card types (for example, only creature cards have values in Power and Toughness), and some variables are lists whose lengths vary from card to card (for example, the list of all sets in which the card was printed). The data was most likely scraped from Gatherer, Magic's official online card index. Therefore the data is accurate and complete, but in a format which Rstudio cannot easily handle.

The Color Identity of a card is often reflected by its mana cost, the price a player must pay to put the card from their hand onto the table, so the mana cost should be ignored in our models as it mostly trivializes the problem. For the same reason we will not search each card's text-box for color-symbols.

Because each of five colors can be either present or absent in a card's Color Identity, there are $2^5 = 32$ possible Color Identities ranging from Colorless (C) to five-colored (WUBRG). It might be troublesome to make a model which selects the correct factor out of thirty-two because of the overlaps in color-philosophies and behavior of multicolored cards, so we resolved to make quintets of models which predict the probability that each of the five colors is present in a card's Color Identity, and then compile the results of these quintets to produce our final guesses. Our error function will therefore use Hamming Distance to compare our predicted Color Identity to the True Color Identity.

Both of our models (based on random forests and logistic regressions) are surprisingly adept at assigning cards Color Identities; both have an average error of less than one color per guess. However, the logistic regression model tends to guess zero-or-one-color Color Identities far more often than it should, making it quite accurate on cards with at most one color in their Color Identity and largely inaccurate on multicolored cards. In conclusion, we consider the random forest model to be more useful.

Data and Methods

```
# install.packages("tidyverse")       # Installation only necessary on the first run
# install.packages("rjson")
# install.packages("gtools")
# install.packages("stringr")
# install.packages("ggplot2")
# install.packages("stringi")
# install.packages("tree")
# install.packages("maptree")
# install.packages("randomForest")
# install.packages("stringdist")
# install.packages("plyr")
# install.packages("ROCR")

library(tidyverse)
library(rjson)      # To read JSON
library(gtools)
library(stringr)    # To interact with strings
library(ggplot2)    # Nice plots
library(stringi)    # Regex parsing
library(tree)
library(maptree)
library(randomForest)
library(stringdist) # Hamming Distance
```

```
library(plyr)       # mapvalues()
library(ROCR)
#setwd('~/Downloads')
setwd("/Users/Theodore/Desktop/R_Studio_Stuff/data")  # Change this to your own directory
MTG.json <- fromJSON(file="AllCards.json",method='C')
    # JSON file from https://mtgjson.com (not great for R)

rows=17761    # When testing, decrease this so you don't use the whole data-set every time

MTG.df <- data.frame(matrix(ncol=15,nrow=rows))   # Start with 15 variables; generate more
```

Here is an example of a typical card in JSON format:

```
MTG.json[["Serra Angel"]]
```

```
## $layout
## [1] "normal"
##
## $name
## [1] "Serra Angel"
##
## $manaCost
## [1] "{3}{W}{W}"
##
## $cmc
## [1] 5
##
## $colors
## [1] "White"
##
## $type
## [1] "Creature <U+0097> Angel"
##
## $types
## [1] "Creature"
##
## $subtypes
## [1] "Angel"
##
## $text
## [1] "Flying (This creature can't be blocked except by creatures with flying or reach.)\nVigilance (A
##
## $power
## [1] "4"
##
## $toughness
## [1] "4"
##
## $imageName
## [1] "serra angel"
##
## $printings
##  [1] "LEA"     "LEB"     "2ED"     "CED"     "CEI"     "3ED"     "4ED"
##  [8] "ATH"     "pWOS"    "7ED"     "8ED"     "9ED"     "10E"     "DDC"
```

```
## [15] "M10"     "M11"     "ME4"     "CMD"     "M12"     "M13"     "M14"
## [22] "M15"     "DD3_DVD" "ORI"     "V15"     "W16"     "EMA"     "W17"
## [29] "CMA"     "IMA"
##
## $legalities
## $legalities[[1]]
## $legalities[[1]]$format
## [1] "Commander"
##
## $legalities[[1]]$legality
## [1] "Legal"
##
##
## $legalities[[2]]
## $legalities[[2]]$format
## [1] "Legacy"
##
## $legalities[[2]]$legality
## [1] "Legal"
##
##
## $legalities[[3]]
## $legalities[[3]]$format
## [1] "Modern"
##
## $legalities[[3]]$legality
## [1] "Legal"
##
##
## $legalities[[4]]
## $legalities[[4]]$format
## [1] "Standard"
##
## $legalities[[4]]$legality
## [1] "Legal"
##
##
## $legalities[[5]]
## $legalities[[5]]$format
## [1] "Vintage"
##
## $legalities[[5]]$legality
## [1] "Legal"
##
##
##
## $colorIdentity
## [1] "W"
```

Clearly this must be parsed to yield useful information in RStudio. Notice how the mana cost of the card reveals its Color Identity, which is why we should ignore mana cost in our models to avoid trivializing the problem. The following chunk should probably be run in a separate script for stability reasons. Here we transform the original data into a more approachable form.

```
     # The function "isItThere" is vital to parsing JSON.
     # Nested lists and inconsistant variable types require it
     # for producing consistant dataframe row-lengths.
     # The argument "json" is a JSON object.
     # "x" is the position in the JSON list to open;
     # each will be a Magic: The Gathering card.
     # "string" is the name of the variable to check for in the JSON element x.
     # "otherwise" is what the program should return
     # if the variable doesn't exist; sometimes "", sometimes NA.
     # "after" is the nested list index in case we need to check
     # elements of a nested list which may not exist.
isItThere <- function(json,x,string,otherwise,after="") {
  if(after!="") {if(is.null(json[[x]][[string]][after])) {return(otherwise)} else
  {return(json[[x]][[string]][after])}} else
    if(is.null(json[[x]][[string]])) {return(otherwise)} else {return(json[[x]][[string]])}}

     # Color-identity is expressed as up to five of the characters WUBR and G.
sapply(1:rows, function(x) {colorIdentity <- c(isItThere(MTG.json,x,"colorIdentity",NA,1),
                                               isItThere(MTG.json,x,"colorIdentity",NA,2),
                                               isItThere(MTG.json,x,"colorIdentity",NA,3),
                                               isItThere(MTG.json,x,"colorIdentity",NA,4),
                                               isItThere(MTG.json,x,"colorIdentity",NA,5))
colorIdentity <- colorIdentity[!is.na(colorIdentity)]

     # Supertype, Type, and Subtype are collections of zero to three character strings
supertype <- c(isItThere(MTG.json,x,"supertypes",NA,1),
               isItThere(MTG.json,x,"supertypes",NA,2),
               isItThere(MTG.json,x,"supertypes",NA,3))
supertype <-supertype[!is.na(supertype)]

type <- c(isItThere(MTG.json,x,"types",NA,1),
          isItThere(MTG.json,x,"types",NA,2),
          isItThere(MTG.json,x,"types",NA,3))
type <-type[!is.na(type)]

subtype <- c(isItThere(MTG.json,x,"subtypes",NA,1),
             isItThere(MTG.json,x,"subtypes",NA,2),
             isItThere(MTG.json,x,"subtypes",NA,3))
subtype <-subtype[!is.na(subtype)]
     # Once these character variables are set, we will decompose them into factors

     # The list of printings is a collection of character-string abbreviations of set-names.
     # We will decompose it into printing dates later.
printList <- c(unlist(lapply(1:length(MTG.json[[x]][["printings"]]),function (y)
                             {MTG.json[[x]][["printings"]][[y]]})))

MTG.df[x,] <<- c(MTG.json[[x]][["name"]],     # We describe all variables as we name them below
                 paste(colorIdentity,collapse=""),
                 MTG.json[[x]][["cmc"]],
                 paste(supertype,collapse=" "),
                 paste(type,collapse=" "),
                 paste(subtype,collapse=" "),
                 isItThere(MTG.json,x,"text",NA),
```

```
                nchar(isItThere(MTG.json,x,"text","")),
                isItThere(MTG.json,x,"power",NA),
                isItThere(MTG.json,x,"toughness",NA),
                isItThere(MTG.json,x,"loyalty",NA),
                length(MTG.json[[x]][["rulings"]]),
                length(MTG.json[[x]][["legalities"]]),
                length(MTG.json[[x]][["printings"]]),
                paste(printList, collapse=" "))})
```

Now we add column names and types:

```
MTG.df <- setNames(MTG.df,c("Name",                 # Character; MTG card name
                            "ColorIdentity",        # Factor; Color Identity
                            "ConvertedManaCost",    # Integer; number of mana required to cast
                            "Supertype",            # Character; broad categories
                            "Type",                 # Character; main card types
                            "Subtype",              # Character; supplimentary types
                            "Text",                 # Character; Rules text, not flavor text
                            "TextLength",           # Integer; number of characters in Text
                            "Power",                # Often an integer, sometimes a character
                            "Toughness",            # Often an integer, sometimes a character
                            "Loyalty",              # Integer; for Planeswalkers
                            "Rulings",              # Number of rulings
                            "Legalities",           # Number of legalities
                            "Printings",            # Number of printings
                            "PrintingsList"))       # Character; all set abbrvs

MTG.df <- transmute(MTG.df, Name=as.character(Name),
                    ColorIdentity=factor(ColorIdentity),
                    ConvertedManaCost=as.integer(ConvertedManaCost),
                    Supertype=as.character(Supertype),
                    Type=as.character(Type),
                    Subtype=as.character(Subtype),
                    Text=as.character(Text),
                    TextLength=as.integer(TextLength),
                    Power=as.character(Power),          # Mostly integers, some odd characters
                    Toughness=as.character(Toughness), # Mostly integers, some odd characters
                    Loyalty=as.integer(Loyalty),
                    Rulings=as.integer(Rulings),
                    Legalities=as.integer(Legalities),
                    Printings=as.integer(Printings),
                    PrintingsList=as.character(PrintingsList))

     # The color identities should be in this order
MTG.df <- mutate(MTG.df,ColorIdentity=factor(ColorIdentity,levels=c("C","W","U","B","R","G",
                "WU","WB","WR","WG","UB","UR","UG","BR","BG","RG","WUB","WUR","WUG","WBR",
                "WBG","WRG","UBR","UBG", "URG","BRG","WUBR", "WUBG","WURG","WBRG",
                "UBRG","WUBRG")))
MTG.df$ColorIdentity[is.na(MTG.df$ColorIdentity)] <- "C"

     # Power and Toughness have some irregularities which should be sorted
MTG.df <- mutate(MTG.df,Power=factor(Power,levels=c("", "*","-1",".5","0","1","1.5","1+*","2",
          "2.5","2+*","3","3.5","4","5","6","7","8","9","10","11","12","13","15","99")))
MTG.df$Power[is.na(MTG.df$Power)] <- ""
```

```
MTG.df <- mutate(MTG.df,Toughness=factor(Toughness,levels=c("", "*","-1",".5","0","1","1.5",
          "1+*","2","2.5","2+*","3","3.5","4","5","6","7","7-*","8","9","10","11","12","13",
          "14","15","99")))
MTG.df$Toughness[is.na(MTG.df$Toughness)] <- ""
```

The supertypes, types, and subtypes of a card may be decomposed into some explanatory factors:

```
MTG.df <- mutate(MTG.df, Legend = str_count(MTG.df$Supertype,"Legendary"),
                 Basic = str_count(MTG.df$Supertype,"Basic"),

                 Artifact = str_count(MTG.df$Type,"Artifact"),
                 Creature =str_count(MTG.df$Type,"Creature")+
                   str_count(MTG.df$Type,"EastureCray"), # A card in a joke set is in pig-latin
                 Enchantment = str_count(MTG.df$Type,"Enchantment"),
                 Land = str_count(MTG.df$Type,"Land"),
                 Instant = str_count(MTG.df$Type,"Instant"),
                 Sorcery = str_count(MTG.df$Type,"Sorcery"),
                 Tribal = str_count(MTG.df$Type,"Tribal"),
                 Planeswalker = str_count(MTG.df$Type,"Planeswalker"))
    # We performed a similar mutation on the subtypes, but it's too lengthy to include.
```

PrintingsList is not a convenient way to check when cards were published. The following mutation is an example of a series of mutations we applied to the data to turn card-set abbreviations into printing years:

```
MTG.df <- mutate(MTG.df, y1993 = as.numeric(str_count(MTG.df$PrintingsList,"LEA") >= 1|
                                            str_count(MTG.df$PrintingsList,"LEB") >= 1|
                                            str_count(MTG.df$PrintingsList,"2ED") >= 1|
                                            str_count(MTG.df$PrintingsList,"ARN") >= 1))
    # The sets Alpha, Beta, Second Edition, and Arabian Nights were printed in 1993.
    # We performed a similar mutation for every year up to 2018.
```

The character variable Text, as it is, is not helpful for algorithmically predicting Color Identity. We decided to count words in Text relevant to each color using Mark Rosewater's "Mechanical Color Pie 2017" as a guide. Below find the summation of words related to the color white as an example:

```
MTG.df <- mutate(MTG.df, WhiteWords= 0+stri_count(MTG.df$Text,regex="(exile(?![^\\(]*\\)))") +
                  stri_count(MTG.df$Text,regex="(prevent(?![^\\(]*\\)))") +
                  stri_count(MTG.df$Text,regex="(defender(?![^\\(]*\\)))") +
                  stri_count(MTG.df$Text,regex="(attacking or blocking(?![^\\(]*\\)))") +
                  stri_count(MTG.df$Text,regex="(blocking or attacking(?![^\\(]*\\)))") +
                  stri_count(MTG.df$Text,regex="(double strike(?![^\\(]*\\)))") +
                  stri_count(MTG.df$Text,regex="(first strike(?![^\\(]*\\)))") +
                  stri_count(MTG.df$Text,regex="(aura(?![^\\(]*\\)))") +
                  stri_count(MTG.df$Text,regex="(enchantment(?![^\\(]*\\)))") +
                  stri_count(MTG.df$Text,regex="(flying(?![^\\(]*\\)))") +
                  stri_count(MTG.df$Text,regex="(indestructible(?![^\\(]*\\)))") +
                  stri_count(MTG.df$Text,regex="(lifelink(?![^\\(]*\\)))") +
                  stri_count(MTG.df$Text,regex="(protection(?![^\\(]*\\)))") +
                  stri_count(MTG.df$Text,regex="(token(?![^\\(]*\\)))") +
                  stri_count(MTG.df$Text,regex="(vigilance(?![^\\(]*\\)))")+
                  stri_count(MTG.df$Text,regex="(destroy all creatures(?![^\\(]*\\)))")+
                  stri_count(MTG.df$Text,regex="(tap target(?![^\\(]*\\)))")+
                  stri_count(MTG.df$Text,regex="(\\-\\d\\/\\+\\d(?![^\\(]*\\)))")+
                  stri_count(MTG.df$Text,regex="(\\+0\\/\\+\\d(?![^\\(]*\\)))")+
                  # Mark Rosewater says the above attributes are primary in White
```

```
                    .5*(stri_count(MTG.df$Text,regex="(destroy target artifact(?![^\\(]*\\)))")+
                    stri_count(MTG.df$Text,regex="(return(?![^\\(]*\\)))")+
                    stri_count(MTG.df$Text,regex="(graveyard(?![^\\(]*\\)))")+
                    stri_count(MTG.df$Text,regex="(scry(?![^\\(]*\\)))")+
                    stri_count(MTG.df$Text,regex="(\\+\\d\\/\\+\\d(?![^\\(]*\\)))")+
                    stri_count(MTG.df$Text,regex="(\\+\\d\\/\\+0(?![^\\(]*\\)))")+
                    # Secondary in white; assign them half a word
                    .15*(stri_count(MTG.df$Text,regex="(each basic land(?![^\\(]*\\)))")+
                    stri_count(MTG.df$Text,regex="(flash(?![^\\(]*\\)))")+
                    stri_count(MTG.df$Text,regex="(hexproof(?![^\\(]*\\)))")+
                    stri_count(MTG.df$Text,regex="(counter(?![^\\(]*\\)))")+
                    stri_count(MTG.df$Text,regex="(reach(?![^\\(]*\\)))")+
                    stri_count(MTG.df$Text,regex="(trample(?![^\\(]*\\)))")+
                    stri_count(MTG.df$Text,regex="(\\-\\d\\/\\-\\d(?![^\\(]*\\)))")))
                    # Tertiary in White; assign .15 of a word
MTG.df$WhiteWords[is.na(MTG.df$WhiteWords)] <- 0
# The regex (blah(?![^(]*\\)) matches all blah outside parentheses.
# This lets the code ignore 'reminder text,' text in parentheses.
# The regex (\\-\\d\\/\\+\\d(?![^\\(]*\\))) matches all -a/+b where a, b are digits
# We performed a similar process with every other color
```

Some of these words are related to more than one color. Nevertheless, they are invaluable in classification. Observe the following:

```
table(MTG.df[MTG.df$WhiteWords>=1,]$ColorIdentity) # Prints the number of cards with
```

```
##
##     C     W     U     B     R     G    WU    WB    WR    WG    UB    UR
##   718  1241   666   689   762   873    75    61    66    83    52    33
##    UG    BR    BG    RG   WUB   WUR   WUG   WBR   WBG   WRG   UBR   UBG
##    33    58    54    57    11     9    12    14     7    11    13     7
##   URG   BRG  WUBR  WUBG  WURG  WBRG  UBRG WUBRG
##     8    12     2     1     0     1     1    11
```

```
                                                    # one white word, by color identity
```

Cards with at least one White word are more likely to have White (W) in their Color Identity, even if the correlation is slight. This suggests that our White keywords are well-chosen for analysing Color Identity.

Now let us divide the data into training and test sets. We randomly pick 7000 observations for training, but add niche values for completeness in model-making.

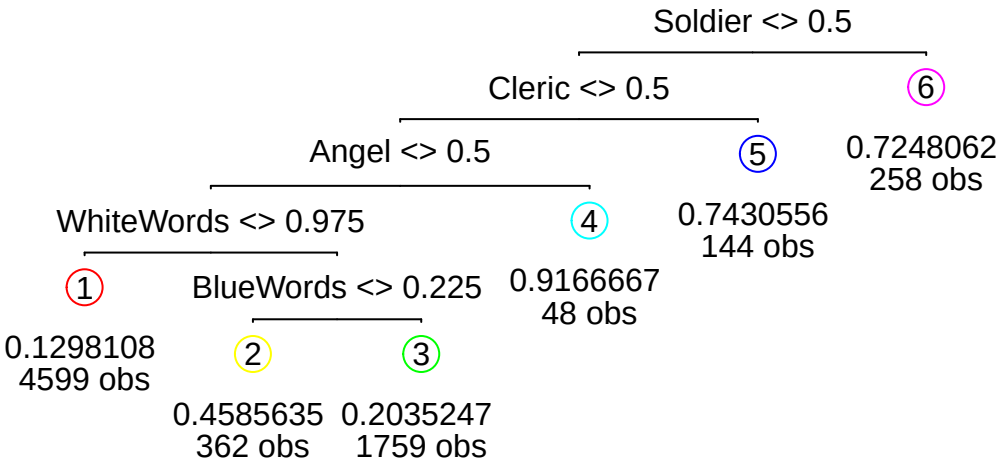
```
rares <- MTG.df[MTG.df$Power %in%
                  c("-1",".5","1.5","1+*","2.5","3.5","11","12","13","15","99"),]
rares <- rbind(rares,MTG.df[MTG.df$Toughness %in%
                  c("-1",".5","1.5","1+*","2.5","3.5","7-*","11","12","13","14","15","99"),])
rares <- rbind(rares,MTG.df[MTG.df$ColorIdentity %in%
                  c("WUBR","WUBG","WURG","WBRG","UBRG","WUBRG","WUR","WBG","UBG"),])
rares <- rbind(rares,MTG.df[MTG.df$ConvertedManaCost %in%
                  c(10,11,12,13,14,15,16,100000),])
rares <- rbind(rares,MTG.df[MTG.df$Loyalty %in%
                  c(2,3,4,5,6,7),])
     # We should make sure our training set includes these niche values

set.seed(999)
train = sample(1:nrow(MTG.df),7000)  # Choose 7000 random cards for training
```

```
MTG.train <- MTG.df[train,]
MTG.train <- rbind(MTG.train,rares)  # Add niche values
MTG.train <- unique(MTG.train)       # Remove duplicates
MTG.test <- MTG.df[-train,]          # Test data includes some niche values from training
```

Before we make a random forest which predicts whether White (W) is in a card's Color Identity, we should perform a 'sanity check' by observing whether a single helpful tree can be made:

```
MTG.White <- select(MTG.train, -c(Name,Supertype,Type,Subtype,Text,PrintingsList,Loyalty,
                        65:90))%>% mutate(ColorIdentity = str_count(ColorIdentity,"W"))
WhiteTree <- tree(ColorIdentity~.,
                  data=MTG.White,control=tree.control(nrow(MTG.train),mindev=.008,minsize=2))
draw.tree(WhiteTree,nodeinfo=FALSE)
```



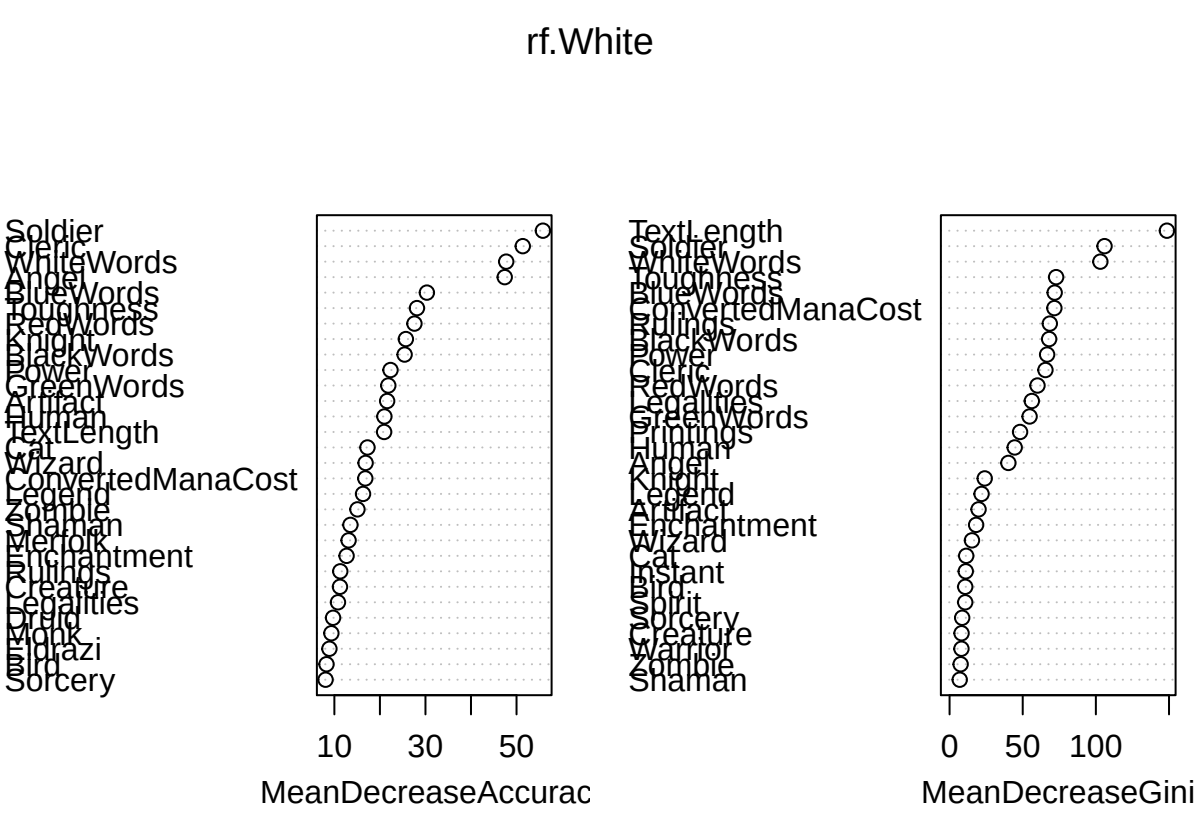
With only a few decisions, we may give a percentage chance a card has White (W) in its Color Identity using some easily understood metrics. For example, if a card is not a soldier, cleric, or angel and it has more than one white word, but less than .225 blue words, we give it a 45.86% percent chance to have White in its Color Identity. We made similar trees for each color, and you can view them in the RMD, but we decided not to include them in the PDF for space considerations. We notice that subtypes are consistantly early splits, suggesting high explanatory power; soldiers are usually white, wizards are usually blue, zombies are usually black, and so on.

Feeling justified in the use of decision trees for Color Identity classification, we set to work defining a quintet of Random Forests.

```
set.seed(1)
MTG.White[sapply(MTG.White, is.character)] <-
  lapply(MTG.White[sapply(MTG.White, is.character)],as.factor)
```

```
White.test <- select(MTG.test,-c(Name,Supertype,Type,Subtype,Text,PrintingsList,Loyalty)) %>%
                      mutate(ColorIdentity = str_count(ColorIdentity,"W"))
White.test[sapply(White.test, is.character)] <-
  lapply(White.test[sapply(White.test, is.character)],as.factor)

rf.White<-randomForest(as.factor(ColorIdentity)~.,data=MTG.White,importance=TRUE)
varImpPlot(rf.White,sort=T)    # Print variable list by impact on method
```



The Mean Decrease in Accuracy generally concurs with the variable choices for splits in the example tree. The Mean Decrease in GINI has some interesting top entries including text-length.

```
print(rf.White)
```

```
##
## Call:
##  randomForest(formula = as.factor(ColorIdentity) ~ ., data = MTG.White,      importance = TRUE)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 7
##
##         OOB estimate of  error rate: 16.05%
## Confusion matrix:
##      0   1 class.error
## 0 5568 143   0.0250394
## 1 1008 451   0.6908842
```

The Out-of-Box Estimate is consistently about 16% across the colors. This is encouraging, suggesting that the forests will perform well on test data.

```
rf.pred=predict(rf.White,White.test)
rf.err = table(pred = rf.pred, truth = White.test$ColorIdentity)
test.rf.err = 1 - sum(diag(rf.err))/sum(rf.err)
test.rf.err # Test error for the random forest
```

```
## [1] 0.1550971
```

```
tree.pred = predict(WhiteTree, White.test)
tree.err=table(pred=tree.pred,truth=White.test$ColorIdentity)
test.tree.err=1 - sum(diag(tree.err))/sum(tree.err)
test.tree.err # Test error for the example tree alone
```

```
## [1] 0.3930861
```

This random forest has much lower test error than the example tree on its own. We made a similar random forest for every color, each one drastically reducing the test error of its related tree.

Now we have five random forests which give probabilities that each of the five colors is present in a card's Color Identity. We may combine their output to predict a card's precise identity.

```
guesses <- data.frame(matrix(ncol=6,nrow=nrow(MTG.train)))
guesses[,1] <- MTG.train$ColorIdentity
guesses[,2] <- predict(rf.White,MTG.White,type="prob")[,2]
guesses[,3] <- predict(rf.blue,MTG.Blue,type="prob")[,2]
guesses[,4] <- predict(rf.Black,MTG.Black,type="prob")[,2]
guesses[,5] <- predict(rf.Red,MTG.Red,type="prob")[,2]
guesses[,6] <- predict(rf.Green,MTG.Green,type="prob")[,2]
     # guesses contains the true training color identities and
     # proportion of trees in each forest declaring a card each color.

guesses <- setNames(guesses,c("ColorIdentity","probWhite","probBlue",
                              "probBlack","probRed","probGreen"))

guesses <- transmute(guesses, ColorIdentity=as.factor(ColorIdentity),
                     probWhite=as.double(probWhite),
                     probBlue=as.double(probBlue),
                     probBlack=as.double(probBlack),
                     probRed=as.double(probRed),
                     probGreen=as.double(probGreen))
guesses$ColorIdentity[guesses$ColorIdentity==""] <- "C"

rf.final<-randomForest(ColorIdentity~.,data=guesses,importance=TRUE)
     # New random forest trained on guesses

final.test <- select(MTG.test,-c(Name,Supertype,Type,Subtype,Text,PrintingsList,Loyalty))
     # Prepare test data

TestGuesses <- data.frame(matrix(ncol=6,nrow=nrow(final.test)))
TestGuesses[,1] <- final.test$ColorIdentity
TestGuesses[,2] <- predict(rf.White,final.test,type="prob")[,2]
TestGuesses[,3] <- predict(rf.blue,final.test,type="prob")[,2]
TestGuesses[,4] <- predict(rf.Black,final.test,type="prob")[,2]
TestGuesses[,5] <- predict(rf.Red,final.test,type="prob")[,2]
TestGuesses[,6] <- predict(rf.Green,final.test,type="prob")[,2]
     # contains the true test color identities and
     # proportion of trees in each forest declaring a card each color.
```

```
TestGuesses <- setNames(TestGuesses,c("ColorIdentity","probWhite","probBlue",
                                "probBlack","probRed","probGreen"))

TestGuesses <- transmute(TestGuesses, ColorIdentity=as.factor(ColorIdentity),
                      probWhite=as.double(probWhite),
                      probBlue=as.double(probBlue),
                      probBlack=as.double(probBlack),
                      probRed=as.double(probRed),
                      probGreen=as.double(probGreen))
TestGuesses$ColorIdentity[TestGuesses$ColorIdentity==""] <- "C"

rf.pred=predict(rf.final,TestGuesses) # Predict test data
rf.err = table(pred = rf.pred, truth = MTG.test$ColorIdentity)
(test.rf.err = 1 - sum(diag(rf.err))/sum(rf.err)) # Test error of the compound model
```

```
## [1] 0.5698355
```

```
rf.err # Large confusion matrix
```

```
##         truth
## pred       C   W   U   B   R   G  WU  WB  WR  WG  UB  UR  UG  BR  BG  RG
##    C     968 202 278 204 227 190  24  16   6  12  31  19   9  31  10  15
##    W      79 798 129  89 135 127  32  19  22  24   3   0   3   5   2  11
##    U      83 135 619 123 138 100  36   6   2   4  25  21  11   5   1   9
##    B      98 109 218 718 151 161  10  18   3  10  40   9   4  25  33   6
##    R      86 116 176 157 669 171   8   8  12   8   8  19   4  28   6  26
##    G      93 133 108 137 178 714   6   6   3  26   4   2  20   7   6  31
##    WU      0  18  19   1   1   1   4   0   0   1   1   0   1   0   0   1
##    WB      4  16   3  19   6   0   2   8   0   1   1   0   0   3   1   0
##    WR      2  34   2   4  21   8   2   2  12   2   0   0   0   0   2   0
##    WG      2  22   2   2   5  33   1   1   0  16   2   0   1   0   0   1
##    UB      8   5  23  36   5   4   4   1   0   1   7   2   1   4   2   3
##    UR      2   1   8   1   5   1   0   1   0   0   0   4   0   0   2   1
##    UG      2   5  17   2   2  25   2   1   1   1   1   1   8   1   1   2
##    BR      4   4   1  24  20   5   1   0   1   0   2   1   0   7   0   1
##    BG      0   3   2  12   6  15   2   1   0   0   0   0   1   1   9   1
##    RG      5   7   9   5  69  42   2   0   2   5   1   1   0   3   2  15
##    WUB     0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
##    WUR     0   1   2   0   1   0   1   0   1   1   0   0   0   0   0   0
##    WUG     0   6   7   0   1   4   0   0   0   0   0   0   1   0   0   0
##    WBR     0   1   0   1   0   0   0   0   0   0   0   0   0   0   0   0
##    WBG     3   7   1   3   0   4   0   0   0   2   0   0   0   0   0   2
##    WRG     0   0   1   0   4   1   1   0   0   0   0   0   1   0   0   1
##    UBR     2   0   1   3   0   0   0   0   0   0   2   1   0   0   0   0
##    UBG     0   0   0   2   0   0   0   0   0   0   1   0   2   0   0   0
##    URG     0   0   0   0   0   2   0   0   0   0   0   0   0   0   0   0
##    BRG     1   2   0   4   4   5   0   0   0   1   0   0   0   3   3   1
##    WUBR    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
##    WUBG    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
##    WURG    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
##    WBRG    0   0   0   0   0   0   0   0   0   1   0   0   0   0   0   0
##    UBRG    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
##    WUBRG   2   8  10   9   6  11   3   0   0   4   4   2   1   2   4   2
##         truth
```

```
## pred    WUB WUR WUG WBR WBG WRG UBR UBG URG BRG WUBR WUBG WURG WBRG UBRG
##   C       6   0   2   1   0   1   3   0   3   1    0    0    0    0    0
##   W       2   0   5   2   0   2   1   0   1   0    0    0    0    0    0
##   U       4   0   3   1   0   0   0   0   0   2    0    0    0    0    0
##   B       4   0   0   2   0   1   7   0   0   4    0    0    0    0    0
##   R       0   0   1   5   0   2   4   0   0   2    0    0    0    0    0
##   G       0   0   1   1   0   7   0   0   1   5    0    0    0    0    0
##   WU      1   0   0   0   0   0   0   0   0   0    0    0    0    0    0
##   WB      1   0   0   0   0   0   0   0   0   0    0    0    0    0    0
##   WR      0   0   0   0   0   0   0   0   0   1    0    0    0    0    0
##   WG      0   0   2   0   0   1   0   0   1   0    0    0    0    0    0
##   UB      1   0   0   0   0   0   5   0   0   0    0    0    0    0    0
##   UR      0   0   0   0   0   0   0   0   1   0    0    0    0    0    0
##   UG      0   0   0   0   0   0   0   0   0   0    0    0    0    0    0
##   BR      0   0   0   1   0   0   3   0   0   3    0    0    0    0    0
##   BG      0   0   0   0   0   0   0   0   0   0    0    0    0    0    0
##   RG      0   0   0   0   0   1   0   0   1   2    0    0    0    0    0
##   WUB     0   0   0   0   0   0   0   0   0   1    0    0    0    0    0
##   WUR     0   9   0   0   0   0   0   0   1   0    0    0    0    0    0
##   WUG     0   0   1   0   0   0   0   0   0   0    0    0    0    0    0
##   WBR     0   0   1   0   0   0   0   0   0   0    0    0    0    0    0
##   WBG     0   0   0   0   6   0   0   0   0   0    0    0    0    0    0
##   WRG     0   0   0   0   0   0   0   0   2   0    0    0    0    0    0
##   UBR     0   0   0   0   0   0   2   0   0   0    0    0    0    0    0
##   UBG     0   0   0   0   0   0   0   6   0   0    0    0    0    0    0
##   URG     0   0   0   0   0   0   0   0   1   0    0    0    0    0    0
##   BRG     0   0   0   0   0   1   0   0   0   0    0    0    0    0    0
##   WUBR    0   0   0   0   0   0   0   0   0   0    0    0    0    0    0
##   WUBG    0   0   0   0   0   0   0   0   0   0    0    1    0    0    0
##   WURG    0   0   0   0   0   0   0   0   0   0    0    0    1    0    0
##   WBRG    0   0   0   0   0   0   0   0   0   0    0    0    0    2    0
##   UBRG    0   0   0   0   0   0   0   0   0   0    0    0    0    0    2
##   WUBRG   2   0   0   1   0   1   1   0   1   2    0    0    0    0    0
##         truth
## pred    WUBRG
##   C         0
##   W         0
##   U         0
##   B         0
##   R         0
##   G         0
##   WU        0
##   WB        0
##   WR        0
##   WG        0
##   UB        0
##   UR        0
##   UG        0
##   BR        0
##   BG        0
##   RG        0
##   WUB       0
##   WUR       0
##   WUG       0
```

```
##   WBR       0
##   WBG       0
##   WRG       0
##   UBR       0
##   UBG       0
##   URG       0
##   BRG       0
##   WUBR      0
##   WUBG      0
##   WURG      0
##   WBRG      0
##   UBRG      0
##   WUBRG    22
```

The test error seems unreasonably high at .569. If given 100 random cards, we would expect only 43 of them to be sorted correctly. However, looking at the confusion matrix, we understand how harshly the algorithm is being measured. If a card's Color Identity is WUB and the random forest suggests WU, it seems unfair to declare its answer totally incorrect. Therefore, we use Hamming distance to assess the difference between our predictions and the actual test values. First we need to write the Color Identities as uniform strings so they may be meaningfully compared:

```
rf.comparison <- data.frame(matrix(ncol=4,nrow=nrow(MTG.test)))
rf.comparison <- setNames(rf.comparison,c("Name","Distance","Predicted","Actual"))

rf.comparison$Name <- MTG.test$Name
rf.comparison$Predicted <- rf.pred
rf.comparison$Actual <- MTG.test$ColorIdentity

rf.comparison[,3] <- mapvalues(rf.comparison[,3],from=c("C","W","U","B","R","G",
"WU","WB","WR","WG","UB","UR","UG","BR","BG","RG","WUB","WUR","WUG","WBR",
"WBG","WRG","UBR","UBG", "URG","BRG","WUBR", "WUBG","WURG","WBRG", "UBRG","WUBRG"),to=c("_____","W____"
                        "WU___","W_B__","W__R_","W___G","_UB__","_U_R_","_U___G","__BR_",
                        "__B_G","___RG","WUB__","WU_R_","WU__G","W_BR_","W_B_G","W__RG","_UBR_",
                        "_UB_G", "_U_RG","__BRG","WUBR_", "WUB_G","WU_RG","W_BRG",
                        "_UBRG","WUBRG"))

rf.comparison[,4] <- mapvalues(rf.comparison[,4],from=c("C","W","U","B","R","G",
"WU","WB","WR","WG","UB","UR","UG","BR","BG","RG","WUB","WUR","WUG","WBR",
"WBG","WRG","UBR","UBG", "URG","BRG","WUBR", "WUBG","WURG","WBRG", "UBRG","WUBRG"),to=c("_____","W____"
                        "WU___","W_B__","W__R_","W___G","_UB__","_U_R_","_U___G","__BR_",
                        "__B_G","___RG","WUB__","WU_R_","WU__G","W_BR_","W_B_G","W__RG","_UBR_",
                        "_UB_G", "_U_RG","__BRG","WUBR_", "WUB_G","WU_RG","W_BRG",
                        "_UBRG","WUBRG"))
```

Now, finding the error:

```
rf.comparison$Distance <- stringdist(rf.comparison[,3],rf.comparison[,4])
sum(rf.comparison$Distance)/nrow(rf.comparison)
```

```
## [1] 0.9491683
```

```
var(rf.comparison$Distance)
```

```
## [1] 0.9116352
```

On average, our quintet of random forests is wrong in its assignment of Color Identity by just under one character. By this metric, the random forest models are a moderate success.

Even though the random forest approach has a relatively high accuracy in predicting color identity of cards, we decided to use another method for comparision. Here we use logistic regressions for color prediction.

First, we build five different fits for each color. There is a summary of logistic regression fit for White cards as an example.

```
set.seed(1)
glm.white = glm(as.factor(ColorIdentity) ~ ., data=MTG.White, family=binomial)
glm.red=glm(as.factor(ColorIdentity) ~ ., data=MTG.Red, family=binomial)
glm.black=glm(as.factor(ColorIdentity) ~ ., data=MTG.Black, family=binomial)
glm.blue=glm(as.factor(ColorIdentity) ~ ., data=MTG.Blue, family=binomial)
glm.green=glm(as.factor(ColorIdentity) ~ ., data=MTG.Green, family=binomial)
summary(glm.white)
```

```
##
## Call:
## glm(formula = as.factor(ColorIdentity) ~ ., family = binomial,
##     data = MTG.White)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
##  -8.49    0.00    0.00    0.00    8.49
##
## Coefficients: (6 not defined because of singularities)
##                      Estimate Std. Error    z value Pr(>|z|)
## (Intercept)       -1.040e+15  4.891e+06 -212573144   <2e-16 ***
## ConvertedManaCost -2.102e+09  6.735e+01  -31211945   <2e-16 ***
## TextLength        -3.298e+11  1.264e+04  -26093638   <2e-16 ***
## Power*             2.953e+15  7.902e+07   37364467   <2e-16 ***
## Power-1            5.214e+14  1.017e+08    5128296   <2e-16 ***
## Power.5           -4.689e+15  6.998e+07  -67004279   <2e-16 ***
## Power0             3.465e+15  7.633e+07   45390292   <2e-16 ***
## Power1             3.763e+15  7.615e+07   49417564   <2e-16 ***
## Power1.5           8.732e+15  1.018e+08   85815157   <2e-16 ***
## Power1+*           3.415e+15  8.680e+07   39349180   <2e-16 ***
## Power2             4.079e+15  7.611e+07   53596630   <2e-16 ***
## Power2.5           1.168e+15  8.968e+07   13025985   <2e-16 ***
## Power2+*          -8.140e+13  6.996e+07   -1163616   <2e-16 ***
## Power3             4.254e+15  7.614e+07   55869391   <2e-16 ***
## Power3.5           2.591e+15  8.650e+07   29956357   <2e-16 ***
## Power4             2.304e+15  7.614e+07   30254164   <2e-16 ***
## Power5             2.618e+15  7.610e+07   34405940   <2e-16 ***
## Power6             2.178e+15  7.629e+07   28550286   <2e-16 ***
## Power7             2.975e+15  7.681e+07   38729754   <2e-16 ***
## Power8             2.914e+15  7.795e+07   37377071   <2e-16 ***
## Power9             2.370e+15  7.101e+07   33378583   <2e-16 ***
## Power10            2.376e+15  7.861e+07   30227440   <2e-16 ***
## Power11            2.043e+15  8.599e+07   23756411   <2e-16 ***
## Power12           -1.665e+15  3.913e+07  -42538736   <2e-16 ***
## Power13            4.847e+15  9.479e+07   51129597   <2e-16 ***
## Power15           -7.900e+14  5.187e+07  -15230655   <2e-16 ***
## Power99           -4.044e+15  6.989e+07  -57867652   <2e-16 ***
## Toughness*        -3.876e+15  7.682e+07  -50455974   <2e-16 ***
## Toughness-1       -4.983e+14  1.207e+08   -4129665   <2e-16 ***
## Toughness.5               NA         NA         NA       NA
```

```
## Toughness0       -5.094e+15  7.418e+07  -68673258   <2e-16 ***
## Toughness1       -5.057e+15  7.344e+07  -68859826   <2e-16 ***
## Toughness1.5     -8.270e+15  9.960e+07  -83039102   <2e-16 ***
## Toughness1+*     -5.225e+15  8.185e+07  -63838885   <2e-16 ***
## Toughness2       -4.740e+15  7.342e+07  -64561320   <2e-16 ***
## Toughness2.5     -6.997e+14  1.102e+08   -6347736   <2e-16 ***
## Toughness2+*             NA         NA         NA       NA
## Toughness3       -4.417e+15  7.341e+07  -60162269   <2e-16 ***
## Toughness3.5     -3.921e+15  8.422e+07  -46554611   <2e-16 ***
## Toughness4       -4.165e+15  7.341e+07  -56730956   <2e-16 ***
## Toughness5       -4.048e+15  7.337e+07  -55165798   <2e-16 ***
## Toughness6       -2.935e+15  7.355e+07  -39902268   <2e-16 ***
## Toughness7       -2.985e+15  7.377e+07  -40459069   <2e-16 ***
## Toughness7-*     -3.563e+15  1.025e+08  -34753085   <2e-16 ***
## Toughness8       -4.278e+15  7.504e+07  -57015748   <2e-16 ***
## Toughness9       -5.912e+15  7.267e+07  -81351614   <2e-16 ***
## Toughness10      -2.289e+15  7.680e+07  -29801830   <2e-16 ***
## Toughness11      -5.510e+15  8.074e+07  -68239793   <2e-16 ***
## Toughness12              NA         NA         NA       NA
## Toughness13      -7.774e+15  8.760e+07  -88748229   <2e-16 ***
## Toughness14              NA         NA         NA       NA
## Toughness15              NA         NA         NA       NA
## Toughness99              NA         NA         NA       NA
## Rulings           5.671e+13  3.720e+05  152442922   <2e-16 ***
## Legalities        2.595e+13  8.293e+05   31286095   <2e-16 ***
## Printings        -2.876e+13  3.633e+05  -79181705   <2e-16 ***
## Legend            6.918e+14  3.972e+06  174162106   <2e-16 ***
## Basic            -1.104e+15  3.524e+07  -31331481   <2e-16 ***
## Artifact         -2.036e+15  4.617e+06 -440989337   <2e-16 ***
## Creature          6.015e+14  1.972e+07   30503377   <2e-16 ***
## Enchantment       5.426e+14  5.203e+06  104287936   <2e-16 ***
## Land             -9.158e+14  6.392e+06 -143270310   <2e-16 ***
## Instant          -3.419e+14  5.252e+06  -65101043   <2e-16 ***
## Sorcery          -7.810e+14  5.273e+06 -148116364   <2e-16 ***
## Tribal           -7.158e+14  1.622e+07  -44123668   <2e-16 ***
## Planeswalker     -4.574e+14  9.231e+06  -49549847   <2e-16 ***
## Artificer        -6.707e+12  1.078e+07    -622348   <2e-16 ***
## Angel             2.884e+15  1.005e+07  286828964   <2e-16 ***
## Aura             -5.300e+14  4.623e+06 -114652035   <2e-16 ***
## Beast            -3.342e+14  6.312e+06  -52942838   <2e-16 ***
## Bird              1.400e+15  7.879e+06  177682109   <2e-16 ***
## Cat               1.342e+15  9.044e+06  148426646   <2e-16 ***
## Cleric            1.499e+15  6.073e+06  246863501   <2e-16 ***
## Construct         6.933e+14  1.048e+07   66153895   <2e-16 ***
## Demon            -4.798e+14  1.077e+07  -44556578   <2e-16 ***
## Dragon            2.878e+14  8.580e+06   33546354   <2e-16 ***
## Giant             1.404e+15  9.998e+06  140443080   <2e-16 ***
## Knight            1.267e+15  8.072e+06  156947672   <2e-16 ***
## Shaman           -1.226e+15  5.996e+06 -204411909   <2e-16 ***
## Druid            -1.524e+15  9.296e+06 -163977036   <2e-16 ***
## Eldrazi          -1.637e+15  9.290e+06 -176219460   <2e-16 ***
## Elemental        -3.257e+14  5.872e+06  -55461797   <2e-16 ***
## Elf              -3.779e+14  6.575e+06  -57477962   <2e-16 ***
## Equipment         2.414e+14  8.127e+06   29705895   <2e-16 ***
```

```
## Golem              2.105e+15  1.102e+07  190957551   <2e-16 ***
## Human              6.265e+14  3.472e+06  180419405   <2e-16 ***
## Horror            -3.689e+14  7.850e+06  -46986606   <2e-16 ***
## Illusion          -2.377e+15  1.267e+07 -187656776   <2e-16 ***
## Insect            -1.058e+15  9.635e+06 -109844651   <2e-16 ***
## Merfolk           -1.033e+15  8.093e+06 -127631948   <2e-16 ***
## Monk               7.728e+14  1.128e+07   68480900   <2e-16 ***
## Rogue             -1.080e+15  7.375e+06 -146457489   <2e-16 ***
## Scout              1.066e+15  9.548e+06  111633264   <2e-16 ***
## Soldier            1.095e+15  4.825e+06  226939816   <2e-16 ***
## Shapeshifter      -1.412e+15  1.160e+07 -121708880   <2e-16 ***
## Spirit             6.626e+14  5.317e+06  124612154   <2e-16 ***
## Sliver             2.327e+14  1.055e+07   22063089   <2e-16 ***
## Snake             -2.667e+15  1.191e+07 -223931501   <2e-16 ***
## Vampire           -6.926e+14  7.618e+06  -90919074   <2e-16 ***
## Warrior           -1.140e+14  4.682e+06  -24351534   <2e-16 ***
## Wall               3.496e+14  1.210e+07   28888225   <2e-16 ***
## Werewolf          -1.303e+15  1.297e+07 -100457864   <2e-16 ***
## Wizard            -4.907e+14  4.963e+06  -98878006   <2e-16 ***
## Wurm              -2.378e+15  1.232e+07 -193078672   <2e-16 ***
## Zombie            -1.210e+15  5.457e+06 -221798537   <2e-16 ***
## WhiteWords         9.594e+14  1.497e+06  640915966   <2e-16 ***
## BlueWords         -3.870e+14  1.031e+06 -375396016   <2e-16 ***
## BlackWords        -6.375e+14  1.266e+06 -503468792   <2e-16 ***
## RedWords          -2.623e+13  1.401e+06  -18716964   <2e-16 ***
## GreenWords         1.283e+14  1.773e+06   72378446   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance:  7244.5  on 7169  degrees of freedom
## Residual deviance: 88234.9  on 7066  degrees of freedom
## AIC: 88443
##
## Number of Fisher Scoring iterations: 25
```

Above we may see which factors are influential in deciding whether a card has white in its color identity or not. For example, cards with the Human subtype have 6.265e+14 added to their score because humans are likely to have White (W) in their Color Identity. Elfs have -3.779e+14 because they are more likely to be Green than White..

In order to evaluate the accuracy of these five logistic regression models, we calculate the test error rate and check the AUC value as well.

```
calc_error_rate <- function(predicted.value, true.value){              # Calculates error
                  return(mean(true.value!=predicted.value))}           #Copy from PSTAT131 HW2's hint
prob.white = predict(glm.white, White.test,type="response")
whitetest=as.factor(ifelse(prob.white<=0.5,0,1)) # try .5 as threshold
print(calc_error_rate(whitetest,White.test$ColorIdentity))
```

```
## [1] 0.1753555
```

```
pred = prediction(prob.white,White.test$ColorIdentity)
performance(pred, "auc")@y.values #AUC for White-card model
```

```
## [[1]]
## [1] 0.6588459
```

```
performance(pred, "auc")@y.values #AUC for Blue-card model
```

```
## [[1]]
## [1] 0.7763734
```

```
performance(pred, "auc")@y.values #AUC for Red-card model
```

```
## [[1]]
## [1] 0.7570065
```

```
performance(pred, "auc")@y.values #AUC for Black-card model
```

```
## [[1]]
## [1] 0.7856732
```

```
performance(pred, "auc")@y.values  #AUC for Green-card model
```

```
## [[1]]
## [1] 0.7858377
```

Comparing five AUC values of logistic models, these are not huge, but they are not discouragingly small. We proceed to made our final model using the outputs of the previous models:

```
set.seed(1)
pred.white<-predict(glm.white,MTG.train,type="response")
pred.blue<-predict(glm.blue,MTG.train,type="response")
pred.black<-predict(glm.black,MTG.train,type="response")
pred.red<-predict(glm.red,MTG.train,type="response")
pred.green<-predict(glm.green,MTG.train,type="response")

guesses2 <- data.frame(matrix(ncol=6,nrow=nrow(MTG.train)))
guesses2[,1] <- MTG.train$ColorIdentity
guesses2[,2] <- exp(pred.white)/(1+exp(pred.white))# stores the probs instead of log(odds)
guesses2[,3] <- exp(pred.blue)/(1+exp(pred.blue))
guesses2[,4] <- exp(pred.black)/(1+exp(pred.black))
guesses2[,5] <- exp(pred.red)/(1+exp(pred.green))
guesses2[,6] <- exp(pred.green)/(1+exp(pred.green))
  #make a table comparing true color identity to the prob from logistic prediction

guesses2 <- setNames(guesses2,c("ColorIdentity","probWhite","probBlue",
                                "probBlack","probRed","probGreen"))

guesses2 <- transmute(guesses2, ColorIdentity=as.factor(ColorIdentity),
                      probWhite=as.double(probWhite),
                      probBlue=as.double(probBlue),
                      probBlack=as.double(probBlack),
                      probRed=as.double(probRed),
                      probGreen=as.double(probGreen))
guesses2$ColorIdentity[guesses2$ColorIdentity==""] <- "C"
```

RStudio warns us that performing prediction with a rank-deficient fit may cause results to be less reliable.

```
library(nnet)
set.seed(1)
glm.final = multinom(as.factor(ColorIdentity) ~ ., guesses2)
```

```
## # weights:  224 (186 variable)
## initial  value 24849.326423
## iter  10 value 15537.767162
## iter  20 value 14564.168362
## iter  30 value 14080.413144
## iter  40 value 13478.334345
## iter  50 value 13174.741207
## iter  60 value 12882.694335
## iter  70 value 12797.390670
## iter  80 value 12759.888031
## iter  90 value 12745.862677
## iter 100 value 12736.744600
## final  value 12736.744600
## stopped after 100 iterations
```

```
  #multiple classification for logistic regression

test.log <- select(MTG.test,-c(Name,Supertype,Type,Subtype,Text,PrintingsList,Loyalty))
test.pred.white<-predict(glm.white,test.log,type="response")
test.pred.blue<-predict(glm.blue,test.log,type="response")
test.pred.black<-predict(glm.black,test.log,type="response")
test.pred.red<- predict(glm.red,test.log,type="response")
test.pred.green<-predict(glm.green,test.log,type="response")

TestGuesses2 <- data.frame(matrix(ncol=6,nrow=nrow(test.log)))
TestGuesses2[,1] <- test.log$ColorIdentity
TestGuesses2[,2] <-exp(test.pred.white)/(1+exp(test.pred.white))#store probs
TestGuesses2[,3] <- exp(test.pred.blue)/(1+exp(test.pred.blue))
TestGuesses2[,4] <- exp(test.pred.black)/(1+exp(test.pred.black))
TestGuesses2[,5] <- exp(test.pred.red)/(1+exp(test.pred.red))
TestGuesses2[,6] <- exp(test.pred.green)/(1+exp(test.pred.green))

TestGuesses2 <- setNames(TestGuesses2,c("ColorIdentity","probWhite","probBlue",
                            "probBlack","probRed","probGreen"))

TestGuesses2 <- transmute(TestGuesses2, ColorIdentity=as.factor(ColorIdentity),
                     probWhite=as.double(probWhite),
                     probBlue=as.double(probBlue),
                     probBlack=as.double(probBlack),
                     probRed=as.double(probRed),
                     probGreen=as.double(probGreen))
TestGuesses2$ColorIdentity[TestGuesses2$ColorIdentity==""] <- "C"

glm.pred = predict(glm.final,TestGuesses2) # Predict test data
glm.err = table(pred = glm.pred, truth = MTG.test$ColorIdentity)
test.glm.err = 1 - sum(diag(glm.err))/sum(glm.err)
test.glm.err # Test error of the compound model
```

```
## [1] 0.5490196
```

```
glm.err #large confusion matrix
```

```
##        truth
## pred       C    W    U    B    R    G   WU   WB   WR   WG   UB   UR   UG
##   C     1146  179   89   91  141   72   32   13    6    5   22   13    3
##   W       35  651  111   80  117  102   35   16   24   33   12    2    4
```

```
##    U        83  359  871  232  313  241   47   20   13   16   31   38   22
##    B        72  180  330  910  322  250    8   26    6   15   56   14    4
##    R        30   72   73  108  453  141    8    3   10    8    3    8    6
##    G        78  191  162  135  308  817   10   10    6   43    8    6   28
##    WU        0    0    0    0    0    0    0    0    0    0    0    0    0
##    WB        0    0    0    0    0    0    0    0    0    0    0    0    0
##    WR        0    0    0    0    0    0    0    0    0    0    0    0    0
##    WG        0    0    0    0    0    0    0    0    0    0    0    0    0
##    UB        0    0    0    0    0    0    0    0    0    0    1    0    0
##    UR        0    0    0    0    0    0    1    0    0    0    0    1    0
##    UG        0    0    0    0    0    0    0    0    0    0    0    0    0
##    BR        0    0    0    0    0    0    0    0    0    0    0    0    0
##    BG        0    0    0    0    0    0    0    0    0    0    0    0    0
##    RG        0    0    0    0    0    0    0    0    0    0    0    0    0
##    WUB       0    0    0    0    0    0    0    0    0    0    0    0    0
##    WUR       0    0    0    0    0    0    0    0    0    0    0    0    0
##    WUG       0    0    0    0    0    0    0    0    0    0    0    0    0
##    WBR       0    0    0    0    0    0    0    0    0    0    0    0    0
##    WBG       0    0    0    0    0    0    0    0    0    0    0    0    0
##    WRG       0    1    0    0    0    0    0    0    0    0    0    0    0
##    UBR       0    0    0    0    0    0    0    0    0    0    0    0    0
##    UBG       0    0    0    0    0    0    0    0    0    0    0    0    0
##    URG       0    0    0    0    0    1    0    0    0    0    0    0    0
##    BRG       0    0    0    0    0    0    0    0    0    0    0    0    0
##    WUBR      0    0    0    0    0    0    0    0    0    0    0    0    0
##    WUBG      0    0    0    0    0    0    0    0    0    0    0    0    0
##    WURG      0    0    0    0    0    0    0    0    0    0    0    0    0
##    WBRG      0    0    0    0    0    0    0    0    0    0    0    0    0
##    UBRG      0    0    0    0    0    0    0    0    0    0    0    0    0
##    WUBRG     0    0    0    0    0    0    0    0    0    0    0    0    1
##          truth
## pred       BR   BG   RG  WUB  WUR  WUG  WBR  WBG  WRG  UBR  UBG  URG  BRG
##    C       32   11   23    7    0    1    0    1    0    3    0    0    3
##    W        4    2   10    3    2    6    4    0    1    2    1    2    0
##    U       15    5   13    9    3    5    3    1    1    4    1    2    0
##    B       39   40   10    1    2    1    1    1    2   15    3    1    7
##    R       20    5   29    0    0    1    6    0    1    1    0    3    4
##    G       15   20   44    1    1    2    0    3   12    0    1    5    9
##    WU       0    0    0    0    0    0    0    0    0    0    0    0    0
##    WB       0    0    0    0    0    0    0    0    0    0    0    0    0
##    WR       0    0    0    0    0    0    0    0    0    0    0    0    0
##    WG       0    0    0    0    0    0    0    0    0    0    0    0    0
##    UB       0    0    0    0    0    0    0    0    0    1    0    0    0
##    UR       0    0    0    0    0    0    0    0    0    0    0    0    0
##    UG       0    0    0    0    0    0    0    0    0    0    0    0    0
##    BR       0    0    0    0    0    0    0    0    0    0    0    0    0
##    BG       0    0    0    0    0    0    0    0    0    0    0    0    0
##    RG       0    0    0    0    0    0    0    0    0    0    0    0    0
##    WUB      0    0    0    0    0    0    0    0    0    0    0    0    0
##    WUR      0    0    0    0    1    0    0    0    0    0    0    0    0
##    WUG      0    0    0    0    0    0    0    0    0    0    0    0    0
##    WBR      0    0    0    0    0    0    0    0    0    0    0    0    0
##    WBG      0    0    0    0    0    0    0    0    0    0    0    0    0
##    WRG      0    0    0    0    0    0    0    0    0    0    0    0    0
```

```
##    UBR      0    0    0    0    0    0    0    0    0    0    0    0    0
##    UBG      0    0    0    0    0    0    0    0    0    0    0    0    0
##    URG      0    0    0    0    0    0    0    0    0    0    0    0    0
##    BRG      0    0    0    0    0    0    0    0    0    0    0    0    0
##    WUBR     0    0    0    0    0    0    0    0    0    0    0    0    0
##    WUBG     0    0    0    0    0    0    0    0    0    0    0    0    0
##    WURG     0    0    0    0    0    0    0    0    0    0    0    0    0
##    WBRG     0    0    0    0    0    0    0    0    0    0    0    0    0
##    UBRG     0    0    0    0    0    0    0    0    0    0    0    0    0
##    WUBRG    0    1    0    0    0    0    0    0    0    0    0    0    0
##          truth
## pred     WUBR WUBG WURG WBRG UBRG WUBRG
##    C        0    0    0    0    0     6
##    W        0    0    1    1    0     3
##    U        0    0    0    0    1     2
##    B        0    0    0    0    0     5
##    R        0    1    0    0    0     1
##    G        0    0    0    1    1     3
##    WU       0    0    0    0    0     0
##    WB       0    0    0    0    0     0
##    WR       0    0    0    0    0     0
##    WG       0    0    0    0    0     0
##    UB       0    0    0    0    0     0
##    UR       0    0    0    0    0     0
##    UG       0    0    0    0    0     0
##    BR       0    0    0    0    0     0
##    BG       0    0    0    0    0     0
##    RG       0    0    0    0    0     0
##    WUB      0    0    0    0    0     0
##    WUR      0    0    0    0    0     0
##    WUG      0    0    0    0    0     0
##    WBR      0    0    0    0    0     0
##    WBG      0    0    0    0    0     0
##    WRG      0    0    0    0    0     0
##    UBR      0    0    0    0    0     0
##    UBG      0    0    0    0    0     0
##    URG      0    0    0    0    0     0
##    BRG      0    0    0    0    0     0
##    WUBR     0    0    0    0    0     0
##    WUBG     0    0    0    0    0     0
##    WURG     0    0    0    0    0     0
##    WBRG     0    0    0    0    0     0
##    UBRG     0    0    0    0    0     0
##    WUBRG    0    0    0    0    0     2
```

The error rate of 0.545 is slightly lower than for the random forests. Given 100 random cards, we would expect the logistic regression to correctly identify 45 of them as opposed to 43. We notice the logistic regression almost always chooses a single color or colorless; the top six rows of the confusion matrix account for most of the predictions. This makes the logistic regression model quite accurate on cards with no color or a single color in their Color Identity but quite inaccurate on multicolored cards.

Besides that, the error rate of 0.535 cannot tell the complete story. As in the random forest approach, it may be too strict when considering mixture colors. Therefore, in order to consider situations where the classifier is partially correct, we use Hamming distance to assess the difference between our predictions and the actual test values for the logistic model as well.

```
library(stringdist)

log.comparison <- data.frame(matrix(ncol=4,nrow=nrow(MTG.test)))
log.comparison <- setNames(log.comparison,c("Name","Distance","Predicted","Actual"))

log.comparison$Name <- MTG.test$Name
log.comparison$Predicted <- glm.pred
log.comparison$Actual <- MTG.test$ColorIdentity

log.comparison[,3] <- mapvalues(log.comparison[,3],from=c("C","W","U","B","R","G",
"WU","WB","WR","WG","UB","UR","UG","BR","BG","RG","WUB","WUR","WUG","WBR",
"WBG","WRG","UBR","UBG", "URG","BRG","WUBR", "WUBG","WURG","WBRG", "UBRG","WUBRG"),to=c("_____","W____"
                        "WU___","W_B__","W__R_","W___G","_UB__","_U_R_","_U___G","__BR_",
                        "__B_G","___RG","WUB__","WU_R_","WU__G","W_BR_","W_B_G","W__RG","_UBR_",
                        "_UB_G", "_U_RG","__BRG","WUBR_", "WUB_G","WU_RG","W_BRG",
                        "_UBRG","WUBRG"))

log.comparison[,4] <- mapvalues(log.comparison[,4],from=c("C","W","U","B","R","G",
"WU","WB","WR","WG","UB","UR","UG","BR","BG","RG","WUB","WUR","WUG","WBR",
"WBG","WRG","UBR","UBG", "URG","BRG","WUBR", "WUBG","WURG","WBRG", "UBRG","WUBRG"),to=c("_____","W____"
                        "WU___","W_B__","W__R_","W___G","_UB__","_U_R_","_U___G","__BR_",
                        "__B_G","___RG","WUB__","WU_R_","WU__G","W_BR_","W_B_G","W__RG","_UBR_",
                        "_UB_G", "_U_RG","__BRG","WUBR_", "WUB_G","WU_RG","W_BRG",
                        "_UBRG","WUBRG"))
log.comparison$Distance <- stringdist(log.comparison[,3],log.comparison[,4])
sum(log.comparison$Distance)/nrow(log.comparison)
```

```
## [1] 0.9949819
```

```
var(log.comparison$Distance)
```

```
## [1] 0.976183
```

The accuracy is comparable to that of the random forest model, possibly because the majority of MTG cards are single-color or Colorless.

Results

We have found two models which predict Magic: The Gathering card Color Identities with an average error of less than one color our of five. What does this really mean? To put it into context, we evaluate the error of a model which always guesses Colorless:

```
sum(stringdist(log.comparison[,4],"_____"))/nrow(MTG.test)
```

```
## [1] 0.9999071
```

```
var(stringdist(log.comparison[,4],"_____"))
```

```
## [1] 0.3242565
```

So even guessing Colorless for every card generates an average error less than one color out of five (with a lower variance!), because the majority of cards have zero, one, or two colors. In this sense our models are unremarkable. However, using the traditional test error rather than Hamming distance, we realize that only the 1444 Colorless cards of the 10761 test cards are properly sorted: the error is .866. Given 100 cards, we would suspect only 13 to be correctly sorted, and these would clearly be coincidences. Our models are

therefore impressive in actually assigning a larger portion of cards correctly than could be cheesed by always choosing any uniform Color Identity.

If we look at a random selection of rows from rf.comparison and log.comparison, it's easy to see that even when our models are not totally accurate they provide some information about a card's Color Identity:

```
set.seed(70)
example <- sample(1:nrow(rf.comparison),10)
print(rf.comparison[example,])
```

```
##                            Name Distance Predicted Actual
## 663           Forbidden Alchemy        2     _____  _UB__
## 10247 Karrthus, Tyrant of Jund        2     WUBRG  __BRG
## 6840                    Buoyancy        0     _U___  _U___
## 1391              Crush Underfoot        0     ___R_  ___R_
## 9297                   Dismantle        2     ____G  ___R_
## 9296               Crazed Goblin        0     ___R_  ___R_
## 3324      Awaken the Sky Tyrant        2     W____  ___R_
## 5348         Blistering Dieflyn        1     ___R_  _____
## 1872      Pentagram of the Ages        0     _____  _____
## 3242         Prismatic Geoscope        0     _____  _____
```

```
print(log.comparison[example,])
```

```
##                            Name Distance Predicted Actual
## 663           Forbidden Alchemy        1     __B__  _UB__
## 10247 Karrthus, Tyrant of Jund        2     __B__  __BRG
## 6840                    Buoyancy        0     _U___  _U___
## 1391              Crush Underfoot        1     _____  ___R_
## 9297                   Dismantle        2     _U___  ___R_
## 9296               Crazed Goblin        0     ___R_  ___R_
## 3324      Awaken the Sky Tyrant        1     _____  ___R_
## 5348         Blistering Dieflyn        0     _____  _____
## 1872      Pentagram of the Ages        0     _____  _____
## 3242         Prismatic Geoscope        0     _____  _____
```

We also see the logistic regression's tendancy to guess single colors or Colorless. It supposes Karrthus, Tyrant of Jund is only Black. The random forests suppose it is five-colored (WUBRG). Both of these guesses are off by Hamming Distance 2 from the actual value, BRG. Therefore we prefer the output of the random forests: the two models have comparable test error measured traditionally and using Hamming Distance, but the random forests are more likely to guess multiple colors, and therefore its guesses communicate more information about the content of the card.

The random forests are also valuable for their splitting variables. By observing the variables which are most important in decision-making, we understand that the subtype of a card is perhaps the most reliable indicator of its color. The forests understand that Elfs are probably Green, and cards which look White but have the subtype Vampire are probably Black or WB. If we analyzed text from the cards' names as well, the forests may have been able to tell that the word "Jund" in Karrthus, Tyrant of Jund signified its BRG Color Identity.

But even the errors in the random forest model provide insight into Magic's design. Magic: The Gathering arranges its colors in a wheel so that every color is adjacent to its two allies and opposite its two enemies: White is allied with Blue and Green, and enemies with Black and Red. When the random forests guess White for a Blue card, we understand the confusion in the overlap of the colors' philosophies.

At the same time, multicolored cards might have access to powers which none of the component colors is associated with. For example, a card which is White/Red (WR) might be able to drain an opponent's life, an ability usually reserved for Black cards. So the forests may be prone to labeling WR cards as B, or vice versa.

Let us check the error rate of the random forest by year:

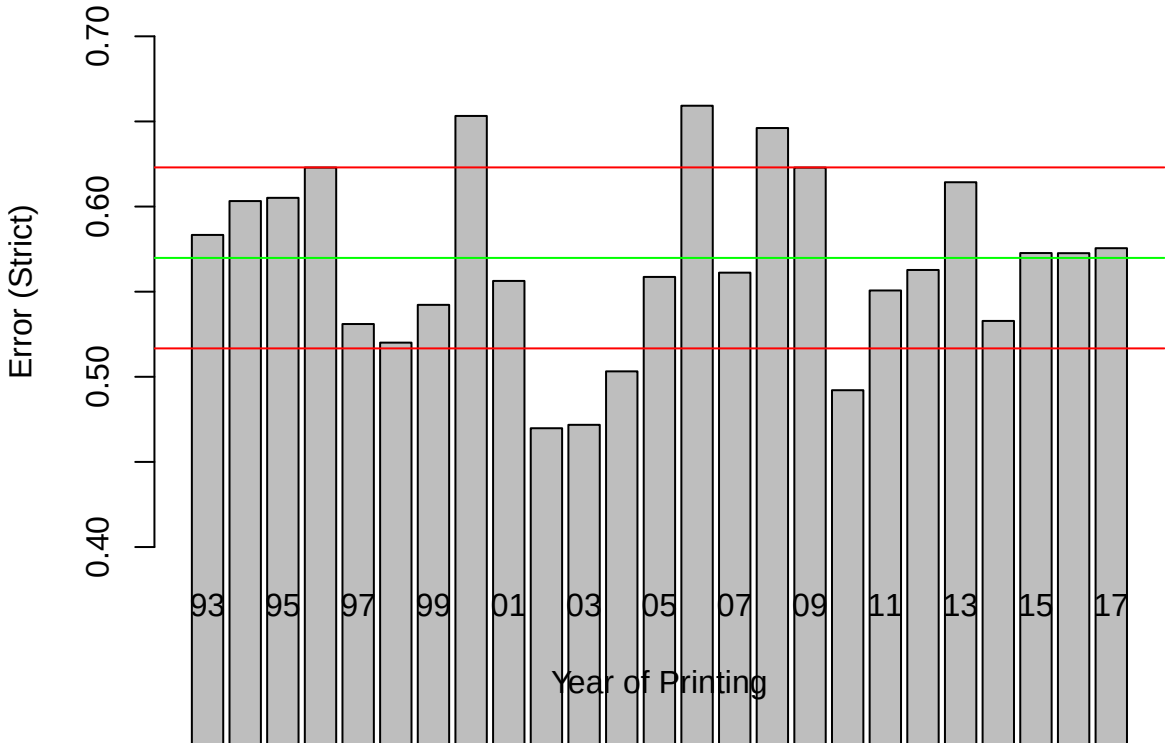
```
test.rf.err.yearly <- data.frame(matrix(ncol=1,nrow=25))

sapply(1:25, function (x)
  {rf.pred <- predict(rf.final,TestGuesses[MTG.test[x+64]==1,])
  rf.err <- table(pred=rf.pred,truth=MTG.test$ColorIdentity[MTG.test[x+64]==1])
  test.rf.err.yearly[x,] <<- 1 - sum(diag(rf.err))/sum(rf.err)})
```

```
##  [1] 0.5833333 0.6032316 0.6051282 0.6228956 0.5310016 0.5200730 0.5422627
##  [8] 0.6532258 0.5563140 0.4697802 0.4717949 0.5031983 0.5586690 0.6592292
## [15] 0.5611702 0.6461318 0.6228869 0.4921260 0.5507088 0.5627530 0.6142857
## [22] 0.5328218 0.5727029 0.5726236 0.5755556
```

```
     # Test error by year of Magic's lifetime

yearPlot <- barplot(test.rf.err.yearly[,1],names.arg=c("93","94","95","96","97","98","99","00","01","02
abline(h=0.5698355,col="Green")      # Test error for whole dataset
abline(h=0.5698355-sqrt(var(test.rf.err.yearly)),col="Red") # Plus 1 SD
abline(h=0.5698355+sqrt(var(test.rf.err.yearly)),col="Red") # Minus 1 SD
```



Test error by year has a variance of .053161. There is no obvious trend to the data, but some years have curiously high or low scores. Some interesting notes:

In 1998 and 2004 the *Unglued* and *Unhinged* sets were released, joke sets where cards have humorous effects. Yet, the test errors in these years are fairly low. This may speak to the robustness of the model, as it correctly sorts these odd-ball cards, or it may suggest overfitting to these odd-ball cards at the expense of ordinary cards.

In 2000 *Prophecy* was considered a poorly-designed set. In the same year *Invasion* was an early block with an emphasis on multicolored cards. Both of these may explain the high error rate in 2000.

2002 saw several sets with unbalanced color proportions. *Torment* was Black-heavy, *Judgement* was Green-and-White-heavy. Perhaps the low test error is related to these broad trends.

2006 had *Dissension*, a multicolor-heavy set, and *TimeSpiral*, a set where cards from previous sets were brought back or referenced. Perhaps these sets explain the difficulty the forests had sorting cards printed this year.

2010 saw *Rise of the Eldrazi* introduce large colorless creatures called the Eldrazi. These powerhouses warped the way the game was played, so cards produced at about this time may not be sortable by the methods suggested by other sets. This is my only explanation for the high test error that year.

Overall these discrepancies are small and random-looking.

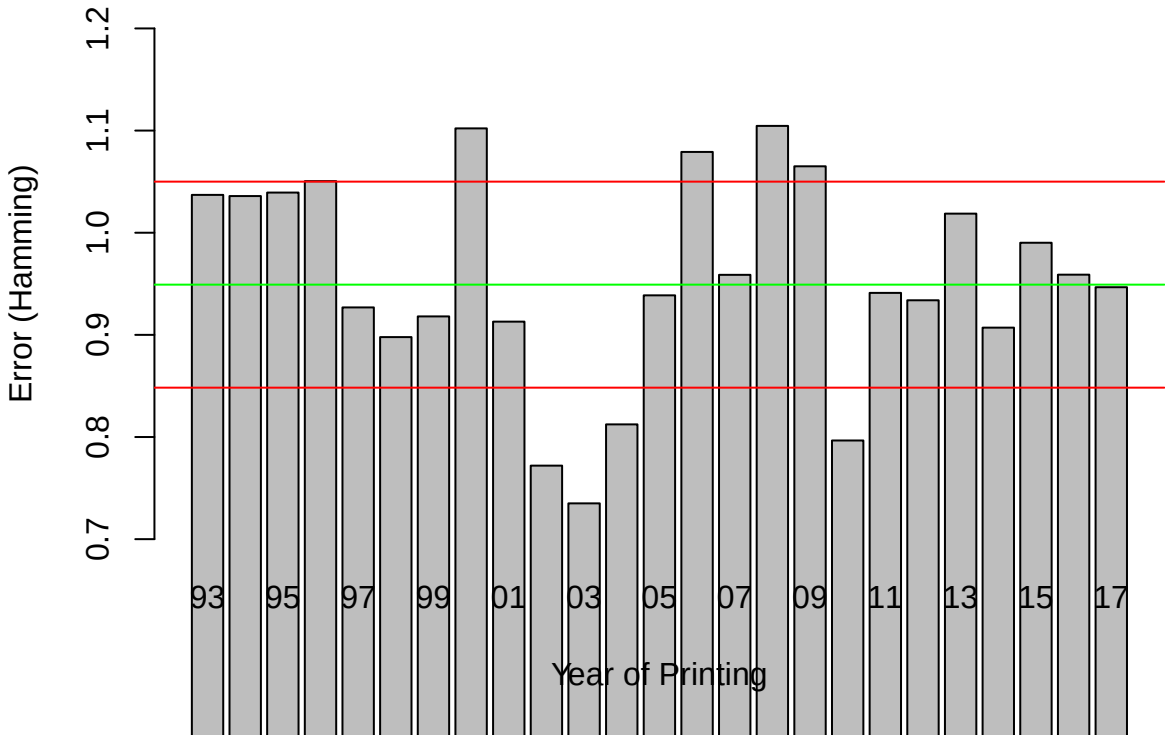
Let us check the same bar plot using Hamming distance:

```
test.rf.err.yearly <- data.frame(matrix(ncol=1,nrow=25))

sapply(1:25, function (x)
  {rf.pred <- predict(rf.final,TestGuesses[MTG.test[x+64]==1,])
  rf.pred <- mapvalues(rf.pred,from=c("C","W","U","B","R","G",
"WU","WB","WR","WG","UB","UR","UG","BR","BG","RG","WUB","WUR","WUG","WBR",
"WBG","WRG","UBR","UBG", "URG","BRG","WUBR", "WUBG","WURG","WBRG", "UBRG","WUBRG"),to=c("_____","W____"
                        "WU___","W_B__","W__R_","W___G","_UB__","_U_R_","_U___G","__BR_",
                        "__B_G","___RG","WUB__","WU_R_","WU__G","W_BR_","W_B_G","W__RG","_UBR_",
                        "_UB_G", "_U_RG","__BRG","WUBR_", "WUB_G","WU_RG","W_BRG",
                        "_UBRG","WUBRG"))
  MTG.test.yearly <- mapvalues(MTG.test$ColorIdentity[MTG.test[x+64]==1],
                               from=c("C","W","U","B","R","G",
"WU","WB","WR","WG","UB","UR","UG","BR","BG","RG","WUB","WUR","WUG","WBR",
"WBG","WRG","UBR","UBG", "URG","BRG","WUBR", "WUBG","WURG","WBRG", "UBRG","WUBRG"),to=c("_____","W____"
                        "WU___","W_B__","W__R_","W___G","_UB__","_U_R_","_U___G","__BR_",
                        "__B_G","___RG","WUB__","WU_R_","WU__G","W_BR_","W_B_G","W__RG","_UBR_",
                        "_UB_G", "_U_RG","__BRG","WUBR_", "WUB_G","WU_RG","W_BRG",
                        "_UBRG","WUBRG"))
  test.rf.err.yearly[x,] <<- sum(stringdist(rf.pred,MTG.test.yearly))/length(rf.pred)})
```

```
##  [1] 1.0370370 1.0359066 1.0393162 1.0505051 0.9268680 0.8978102 0.9180754
##  [8] 1.1021505 0.9129693 0.7719780 0.7350427 0.8123667 0.9387040 1.0791075
## [15] 0.9587766 1.1045845 1.0650195 0.7965879 0.9411123 0.9338731 1.0186813
## [22] 0.9070759 0.9901873 0.9589354 0.9466667
```

```
yearPlot <- barplot(test.rf.err.yearly[,1],names.arg=c("93","94","95","96","97","98","99","00","01","02
abline(h=0.9491683,col="Green")
abline(h=0.9491683-sqrt(var(test.rf.err.yearly)),col="Red")
abline(h=0.9491683+sqrt(var(test.rf.err.yearly)),col="Red")
```



This is too similar to the previous plot to say much, except justify Hamming Distance as a measurement of correctness.

Conclusions

Although we were able to algorithmically sort Magic: The Gathering cards by Color-Identity with relative reliability, there is much room for improvement. The whole set of Magic cards presents uniquely complicated problems in data-mining. There are cards which are double-sided, and even cards which have two miniature cards printed on the same side. When a card with Power or Toughness "/" or "X/X" is to be sorted, the method for choosing * or X should be considered. Some keywords, such as "counter," can mean different things depending on context (it may mean to counter a spell, or it may mean to place a counter on a spell) and the different meanings are related to different colors. Artifacts are usually Colorless, but exceptions exist. Perhaps no perfect algorithm exists for guessing Color Identity.

Even the Magic: The Gathering designers can get it wrong. Magic's lead designer Mark Rosewater laments the decisions made in a set called "Planar Shift" where they mixed color-philosophies in non-traditional ways. Even considering this, Magic's design has changed over its lifetime. Abilities once considered taboo in Red like card-drawing have been reconsidered. The designers find interesting ways to allow cards of different colors to perform effects in a way consistant with their philosophies.

Knowing this, maybe our analytic approach could be helpful in assigning Color Identities in future sets. This could be used to enforce a consistent feeling for each Color Identity.

References

Schults, Robert. "Provides Magic: the Gathering Card Data in JSON Format." *MTGJSON*, 13 Nov. 2017, mtgjson.com/.

Rosewater, Mark. "Mechanical Color Pie 2017." *MAGIC : THE GATHERING*, Wizards of the Coast, 5 June 2017, magic.wizards.com/en/articles/archive/making-magic/mechanical-color-pie-2017-2017-06-05.

"RoboRosewater (@RoboRosewater)." *Twitter*, 3 July 2015, twitter.com/RoboRosewater?ref_src=twsrc%5Egoogle%7Ctwcamp%5Eserp%7Ctwgr%5Eauthor.

"Gatherer." *Magic : The Gathering*, Wizards of the Coast, gatherer.wizards.com/Pages/Default.aspx.