# Intro To R Final Project

*Conor Falvey, Tanja Neundel, Mushahid Hassan*

*10/31/2019*

Our motivation for this project came from all three of us sharing a common interest in machine learning. Since all three of us come from different backgrounds and speak multiple languages, we had to establish an affective way to communicate when deciding what our project was going to be. During our meetings, we started to notice how similar some English words are to some of the words in our respective languages. This then steered us into natural language processing using machine learning as we wanted to learn more about the origins of the English language.

The data thst we analyze is from 6 different text files that we custom created by referencing off of this. Each file contains a
list of English words that derive from a certain language. The name of the language is the file name. For example the file, "anglo.txt" contains a list of English words that originate from Anglo-Saxon or, Old English. "french.txt" contains English words originating from French, "latin.txt" from Latin, and etc. The file "10k.txt" contains the 10,000 most used words in the current English language and those are the words the neural network operates on.

The biggest problem we face by far, is trying to generalize a single word into just one category. There are plenty of words in the English language the originate from multiple languages as opposed to just one. Although we can make our neural network to take into account all the different languages a word might originate from, there are just too many outlying factors to take care of that can make our inference 100% correct and free from error.

Packages we used for this project:

```r
knitr::opts_chunk$set(echo = TRUE)
library(class)
library(neuralnet)
library(gtools)
library(stringr)
library(SnowballC)
```

One of the main things we learned from this project is how powerful neural networks can be. They're not the most friendly to work with, however, they epitomize the whole concept of machine learning. Starting off by training a computer to recognize small patterns, we can quickly create an selection of different possibilities that can lead to some really interesting results. Neural networks are also useful in a sense that they can give a fairly accurate result even when given sub-par knowledge because they build upon prior inferences.
One thing we could improve about this project is have more accurate and fixed data sets. Since we created our own datasets from a wikipedia page, it's not the best. However, that is one of the struggles when evaluating the origins of a language, there's only a handful of information you can find online, becuase language itself is so complex. Thus, we had to settle for creating our own datasets.

## Label Data

```r
words <- readLines("./10k.txt")
anglo <- readLines("./anglo.txt")
norse <- readLines("./oldnorse.txt")
```

```r
french <- readLines("./french.txt")
latin <- readLines("./latin.txt")

#Set column names and find words with no known origin
labels <- c("word", "ety")
words <- setdiff(setdiff(setdiff(setdiff(words, anglo), norse), french), latin)

#Label data with distinct numbers
no.label <- data.frame(words, rep(0, length(words)))
anglo.df <- data.frame(anglo, rep(1, length(anglo)))
norse.df <- data.frame(norse, rep(2, length(norse)))
french.df <- data.frame(french, rep(3, length(french)))
latin.df <- data.frame(latin, rep(4, length(latin)))

names(no.label) <- labels
names(anglo.df) <- labels
names(norse.df) <- labels
names(french.df) <- labels
names(latin.df) <- labels

#Join into one labelled dataframe
data <- rbind(rbind(rbind(rbind(no.label, anglo.df), norse.df), french.df), latin.df)
data[, 1] <- tolower(data[, 1])
data <- data[-1, ]
```

## Getting Statistics

```r
#Developing secondary statistics
for (i in 1:length(data$word)) {
  data[i, 3] <- sum(asc(data[i, 1])) / str_length(data[i, 1])
  data[i, 4] <- wordStem(data[i, 1], language = "english")
  data[i, 5] <- sum(asc(data[i, 4])) / str_length(data[i, 4])
  data[i, 6] <- data[i, 5] - data[i, 3]
}

labels <- c("word", "ety", "score", "stem", "stemScore", "diff")
names(data) <- labels

numerics <- data.frame(data[, 2], data[, 3], data[, 5], data[, 6])
names(numerics) <- c("ety", "score", "stemScore", "diff")
```

## Planning Neural Network

```r
#Split dataset for testing and training
index <- sample(1:nrow(numerics),round(0.75*nrow(numerics)))
train <- numerics[index,]
test <- numerics[-index,]

#Create linear model to test error against
```

```r
lm.fit <- glm(train$ety ~ ., data=train)
summary(lm.fit)
```

```
##
## Call:
## glm(formula = train$ety ~ ., data = train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -5.5257  -1.7464   0.9666   1.1123   2.3539
##
## Coefficients: (1 not defined because of singularities)
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -3.69556    0.40989  -9.016  < 2e-16 ***
## score        0.08348    0.01146   7.287 3.32e-13 ***
## stemScore   -0.02197    0.01081  -2.033   0.0421 *
## diff              NA         NA      NA       NA
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 2.509388)
##
##     Null deviance: 41900  on 16435  degrees of freedom
## Residual deviance: 41237  on 16433  degrees of freedom
## AIC: 61770
##
## Number of Fisher Scoring iterations: 2
```

```r
pr.lm <- predict(lm.fit,test)
```

```
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type
## == : prediction from a rank-deficient fit may be misleading
```

```r
MSE.lm <- sum((pr.lm - test$ety)^2)/nrow(test)

#Scale data to standardized input set
maxs <- apply(numerics, 2, max)
mins <- apply(numerics, 2, min)

scaled <- as.data.frame(scale(numerics, center = mins, scale = maxs - mins))

train_ <- scaled[index,]
test_ <- scaled[-index,]
```

## Running Network

```r
# Set names of columns in neural network
n <- names(train_)

# Create formula for use in network
```

```r
f <- as.formula(paste("train_$ety ~", paste(n[!n %in% "ety"], collapse = " + ")))

# Run the network!
nn <- neuralnet(f, data = train_, hidden = c(5, 5, 3, 3), linear.output = T,
                threshold = 0.1, stepmax = 1e6)

# Test with testing set
pr.nn <- compute(nn, test_[ ,1:4])

# Compute the error of the network
pr.nn_ <- pr.nn$net.result*(max(numerics$ety)-min(numerics$ety))+min(numerics$ety)
test.r <- (test_$ety)*(max(numerics$ety)-min(numerics$ety))+min(numerics$ety)

MSE.nn <- sum((test.r - pr.nn_)^2)/nrow(test_)

# Compare how effective it is versus a linear model
print(paste(MSE.lm,MSE.nn))
```

```
## [1] "2.5133619937486 2.47707871746587"
```

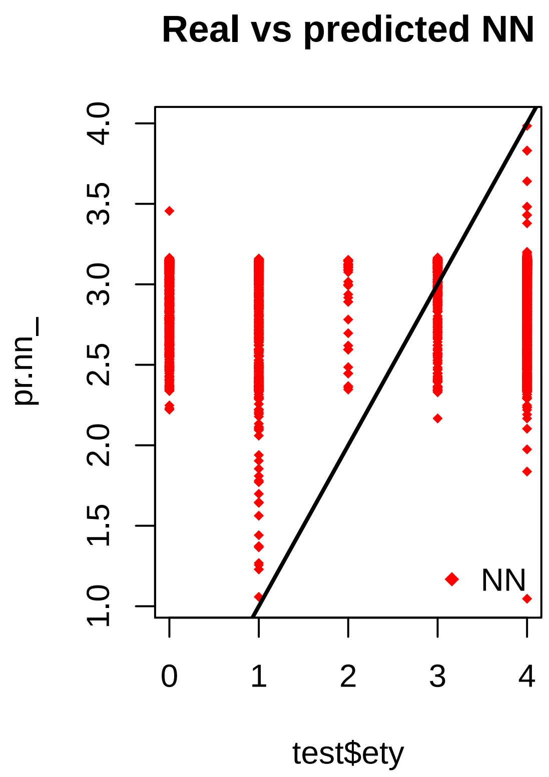
## Function wrapper

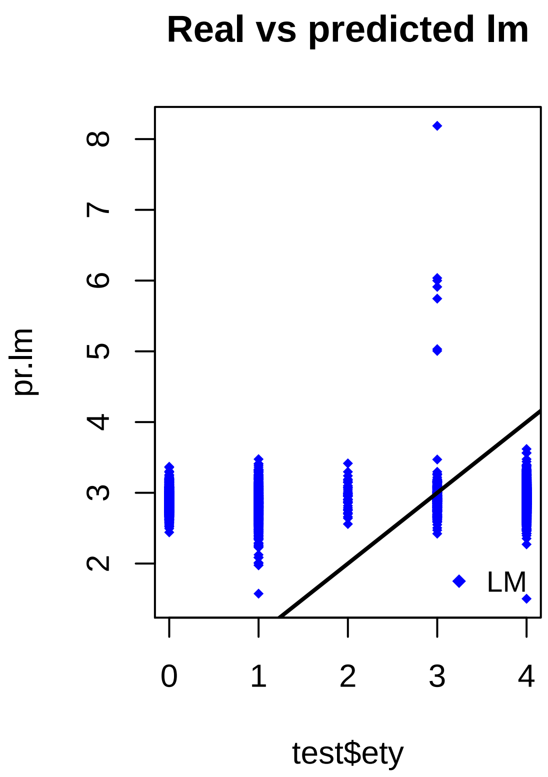
```r
testNetwork <- function(word) {
  score <- sum(asc(word)) / str_length(word)
  stemScore <- sum(asc(wordStem(word, language = "english"))) /
    str_length(wordStem(word, language = "english"))
  difference <- stemScore - score

  tester <- data.frame(word, score, stemScore, difference)
  names(tester) <- c("ety", "score", "stemScore", "diff")

  comp <- compute(nn, tester)
  print(comp$net.result * 4)
}
testNetwork("random")
```

```
##          [,1]
## [1,] 3.079554
```

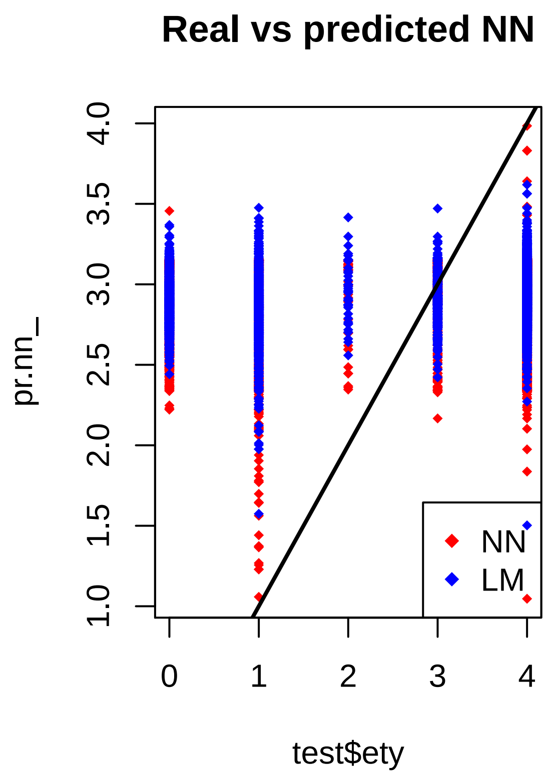
#Visualization

```r
par(mfrow=c(1,2))

plot(test$ety, pr.nn_, col = 'red', main = 'Real vs predicted NN', pch = 18, cex = 0.7)
abline(0, 1, lwd = 2)
legend('bottomright', legend = 'NN', pch = 18, col = 'red', bty = 'n')

plot(test$ety, pr.lm, col = 'blue', main = 'Real vs predicted lm', pch = 18, cex = 0.7)
abline(0, 1, lwd = 2)
legend('bottomright', legend = 'LM', pch = 18, col = 'blue', bty = 'n', cex = .95)
```





```r
plot(test$ety, pr.nn_, col = 'red', main = 'Real vs predicted NN', pch = 18, cex = 0.7)
points(test$ety, pr.lm, col = 'blue', pch = 18, cex = 0.7)
abline(0, 1, lwd = 2)
legend('bottomright', legend = c('NN','LM'), pch = 18, col = c('red','blue'))
```


Real vs predicted NN
pr.nn_
test$ety
1.0
1.5
2.0
2.5
3.0
3.5
4.0
0
1
2
3
4
◆ NN
◆ LM