Pro Data Scientists - Cyberbullying Classifier

Contents

Team members

Leon Harper (21385662)

Thomas Newton (21365654)

Michal Jedruszczak (21440496)

These are the team members for the group project.

Step 1: Problem statement

World of Warcraft Problem Statement

World of Warcraft is a popular MMORPG (Massively multiplayer online role-playing game) video game with millions of user every month. Due to having this many players there is bound to be some cyberbullying/toxic players included in those millions of players, World of Warcraft however is especially toxic and is often ranked as one of the most toxic gaming community's today. In a survey by ADL (Anti-Defamation League) it was found 66% of adults ages 18-45 have been harassed/bullied in World of Warcraft in 2021 (Hate is No Game: Harassment and Positive Social Experiences in Online Games 2021).

The objective of this project is to create a model that will be able to detect cyberbullying/toxicity. By using the World of Warcraft dataset provided to us it will allow the model to have reference for comments deemed as bullying.

This would be a classification model that when give a comment/statement would decided whether it is bullying or not bullying, it will be able to do this by detecting certain words and phrases.

Currently most video games have an option to filter chat, however this only censors certain words/phrases. Our model will be able to detect strings of words rather than just certain ones.

League of Legends problem statement

League of Legends is a popular MOBA (Multiplayer online battle arena) video game with millions of users every month. Due to having this many players there is bound to be some cyberbullying/toxic players included in those millions of players, League however is especially toxic and is often ranked as one of the most toxic gaming community's today. The current ban rate of all accounts as of September 2022 is 2.25% (Around 2,632,500 account). In a survey by ADL (Anti-Defamation League) it was found 65% of adults ages 18-45 have been harassed/bullied in League of Legends in 2021.

The objective of this project is to create a model that will be able to detect cyberbullying/toxicity. By using the League of legends dataset provided to us it will allow the model to have reference for comments deemed as bullying.

This would be a classification model that when give a comment/statement would decided whether it is bullying or not bullying, it will be able to do this by detecting certain words and phrases.

Currently most video games have an option to filter chat, however this only censors certain words/phrases. Our model will be able to detect strings of words rather than just certain ones.

ADL Survey: Hate is No Game: Harassment and Positive Social Experiences in Online Games 2021 (adl.org)
Step 2: Importing data

```
wow_posts_df <- read.csv("Data/posts_wow.csv")
wow_annotations_df <- read.csv("Data/annotations_wow.csv")

lol_posts_df <- read.csv("Data/posts_lol.csv")
lol_annotations_df <- read.csv("Data/annotations_lol.csv")
```

This imports the required data for the project. The data was exported from an SQL script that creates the necessary tables (i.e. posts and annotations) and the data. To simplify the process of importing data, we used the table export wizard to export the SQL table data into csv files using custom SQL as the MySQL Workbench Table Export Wizard doesn't export all of the data properly.

Step 2.5: Preliminary EDA

We are doing a preliminary EDA in order to understand how we should clean the data and the kind of data that we are dealing with.

League of Legends analysis

```
summary(lol_posts_df)
```

```
##     topic_id       post_number      author          html_message
##  Min.   :   2.0   Min.   :   0   Length:16867       Length:16867
##  1st Qu.:  33.0   1st Qu.: 364   Class :character   Class :character
```

```
##  Median : 140.0   Median : 953   Mode  :character   Mode  :character
##  Mean   : 604.6   Mean   :1393
##  3rd Qu.:1090.0   3rd Qu.:1998
##  Max.   :2481.0   Max.   :5358
##   timestamp
##  Length:16867
##  Class :character
##  Mode  :character
##
##
##
```

```
nrow(lol_posts_df)
```

```
## [1] 16867
```

There are 16867 data points in the League of Legends posts data.

```
lol_posts_df %>% select("topic_id") -> lol_topics #gets all unique lol_topics from the dataset
nrow(unique(lol_topics))
```

```
## [1] 17
```

This data set has a total of 17 different topics.

World of Warcraft analysis

In our World of Warcraft posts dataset, we are given 16978 rows of 5 features: topic_id, post_number, author, html_message and timestamp. For the purposes of this project, we will ignore timestamp, as it will not be used to train our models. The additional WoW annotations dataset reveals which messages were flagged as cyberbullying, from which the only useful features are the post_number and topic_id. This information has been combined into a single file – clean_posts_balanced_sample.csv.

Posts in the main dataset are formatted using HTML, meaning that our model will either have to be trained to recognise patterns such as paragraph breaks or we will have to clean the data and transform it into something more appropriate.

Furthermore, as we are only using a single set of features (post_number and topic_id) from the annotations dataset, we may be able to add another column to the posts dataset – a Boolean value representing whether or not a specific post contains cyberbullying. This will eliminate the need for the use of two separate datasets to develop our models.

Regarding the types of data we're given, topic IDs, authors and HTML messages are categorical, while post numbers are ordinal. In the annotations data set, all values but post number are categorical.

```
summary(wow_posts_df)
```

```
##    topic_id          post_number      author          html_message
##  Length:16978       Min.   :   0   Length:16978       Length:16978
##  Class :character   1st Qu.: 226   Class :character   Class :character
##  Mode  :character   Median : 708   Mode  :character   Mode  :character
##                     Mean   :1366
```

```
##                     3rd Qu.:2420
##                     Max.   :4692
##                     NA's   :3
##   timestamp
##  Length:16978
##  Class :character
##  Mode  :character
##
##
##
##
```

```
nrow(wow_posts_df)
```

```
## [1] 16978
```

There are 16978 data points.

```
wow_posts_df %>% select("topic_id") -> wow_topics #gets all unique wow_topics from the dataset
nrow(unique(wow_topics))
```

```
## [1] 23
```

This data set has a total of 23 different topics.

Step 3: Cleaning the data / Pre-processing

For pre-processing, we decided to merge the datasets as they are essentially identical to each other in terms of structure and we believed that this would simplify data pre-processing and remove code duplication. We also ignored the annotation csv files (apart from feature extraction) as we found that there wasn't anything to clean (we would be just cleaning metadata). As we merged the datasets, we found that we could create a "is_bullying" column that would be used as a target variable for the model building process. We also created a "bullying_severity" column as each post can have multiple annotators for each cyberbullying post.

After that, we extracted the messages out of the HTML in order to then pre-process the messages through stemming, removing punctuation and removing stopwords (the messages are HTML as both datasets come from web forums). We then extracted a "word_counts" column for EDA purposes as well as sampling the data in order to balance it out. We then created training and test datasets for model building where each dataset is a document term matrix as we cannot pass in raw text data to the models. Finally, we export the relevant data (including clean data).

```
# Creates dataset column to merge posts and annotations csv files together
wow_posts_df$dataset <- "WoW"
lol_posts_df$dataset <- "LoL"
wow_annotations_df$dataset <- "WoW"
lol_annotations_df$dataset <- "LoL"

posts_df <- rbind(wow_posts_df, lol_posts_df)
annotations_df <- rbind(lol_annotations_df, wow_annotations_df)
```

Since wow_posts_df and lol_posts_df have the same structure, we merged the posts and annotation data frames together to simplify pre-processing (this avoids repeating code). However, we will need to analyse the datasets separately for EDA purposes so we created a "dataset" feature to counteract this.

```
posts_df$id <- paste(posts_df$dataset, posts_df$topic_id, posts_df$post_number, sep="_")
annotations_df$id <- paste(annotations_df$dataset, annotations_df$topic_id, annotations_df$post_number,
```

To simplify the merging of data frames, we will create an ID column so that a left join can be performed on a single column. This mitigates the issues of duplicate topic ids and post numbers as the post numbers are only unique according to the topic id.

```
merged_df <- left_join(posts_df, annotations_df, by = "id", keep=TRUE)
merged_df$is_bullying <- as.integer(!is.na(merged_df["id.y"]))
drop <- c('topic_id.y', 'post_number.y', 'dataset.y', 'id.y', 'offender', 'victim')
merged_df <- merged_df[, !(names(merged_df) %in% drop)]

# Removes the ".x" characters from the remaining annotations columns
colnames(merged_df) = sub(".x", "", colnames(merged_df))

# Create bullying_severity column
names(merged_df)[names(merged_df) == "annotator"] <- "bullying_severity"
merged_df["bullying_severity"][is.na(merged_df["bullying_severity"])] <- 0
posts_df <- merged_df %>% group_by(id) %>% slice(which.max(bullying_severity))
```

This code performs a left join to merge the dataframes together. Most of the columns from the annotations dataframe are useless for training an NLP classifier so we will be removing those columns. Since there are duplicate columns on each side, we will be dropping "y" columns.

We also created a bullying_severity column as we found that some posts have been annotated as bullying by multiple annotators which could make this a useful feature for model building.

```
remove_html <- function(html_msg, isHtml) {
  if(isHtml) {
    # Remove backslashes when dealing with LoL forum data
    html_msg <- gsub("\\\\", '', html_msg)
    # Get XML nodes
    msg <- xml2::read_html(html_msg)
    # Get the block quotes and quotes (blockquotes for WoW, .quote for LoL)
    blockquotes <- msg %>% html_nodes("blockquote")
    quotes <- msg %>% html_nodes(".quote")

    # Remove quote elements for LoL and WoW datasets
    xml_remove(blockquotes)
    xml_remove(quotes)
    msg <- html_text(msg)
    return(msg)
  }
  return(html_msg)
}
```

The "html_message" column has messages that have HTML and do not contain HTML at all. In order to handle this, we will be creating a "is_html" column that uses a regular expression to detect HTML in order to prevent errors with RVest. The "tm" package does not handle removing HTML content and we cannot simply use a regular expression to remove HTML as the data originates from gaming forums where "

” elements are frequently used. If we used a regular expression then the content inside the blockquotes would still remain.

To remove the content of the blockquotes, we used RVest to acquire the blockquote element contents as well as any

elements with “.quote” and then we use xml_remove() to remove the blockquote element nodes. We then convert the RVest object back into a string.

```
# Regex for detecting HTML
detect_html_regex <- "<.*?>"
# Create is_html column
posts_df$is_html <- str_detect(posts_df$html_message, detect_html_regex)
# Apply remove_html function to html_message
posts_df$html_message <- mapply(remove_html, posts_df$html_message, posts_df$is_html)
posts_df <- posts_df[, !(names(posts_df) %in% 'is_html')]

# Converts any regex passed into the transformer into a space character
toSpaceTransformer <- content_transformer(function (x, pattern) gsub(pattern, "", x))
posts_corpus <- Corpus(VectorSource(posts_df$html_message))
posts_corpus <- posts_corpus %>%
                tm_map(content_transformer(tolower)) %>%
                tm_map(toSpaceTransformer, "http\\S+\\s*") %>%
                tm_map(removeNumbers) %>%
                tm_map(removeWords, stopwords("english")) %>%
                tm_map(removePunctuation) %>%
                tm_map(stemDocument) %>%
                tm_map(stripWhitespace)
posts_df$html_message <- data.frame(text=sapply(posts_corpus, identity), stringsAsFactors = F)$text
```

This code removes useless characters, stopwords, punctuation and it uses stemming to improve model performance. Certain steps of the pre-processing could be tweaked to improve model performance (e.g. number of stopwords being omitted) as the pre-processing could end up being too rigorous.

We removed the HTML characters first in order to prevent interference when removing punctuation or whitespace.

```
posts_df$word_counts <- str_count(posts_df$html_message, "\\S+")
```

This code gets the word counts for the html messages which can be used for analysing word counts in the EDA. We may also use the word counts to filter messages with word counts that are too low.

```
posts_df <- posts_df %>% na_if("") %>% na.omit
```

This code removes NaN rows from posts_df which can become a problem after pre-processing if there were too many stop words in the original messages.

```
write.csv(posts_df, file="Data/clean_posts.csv")
```

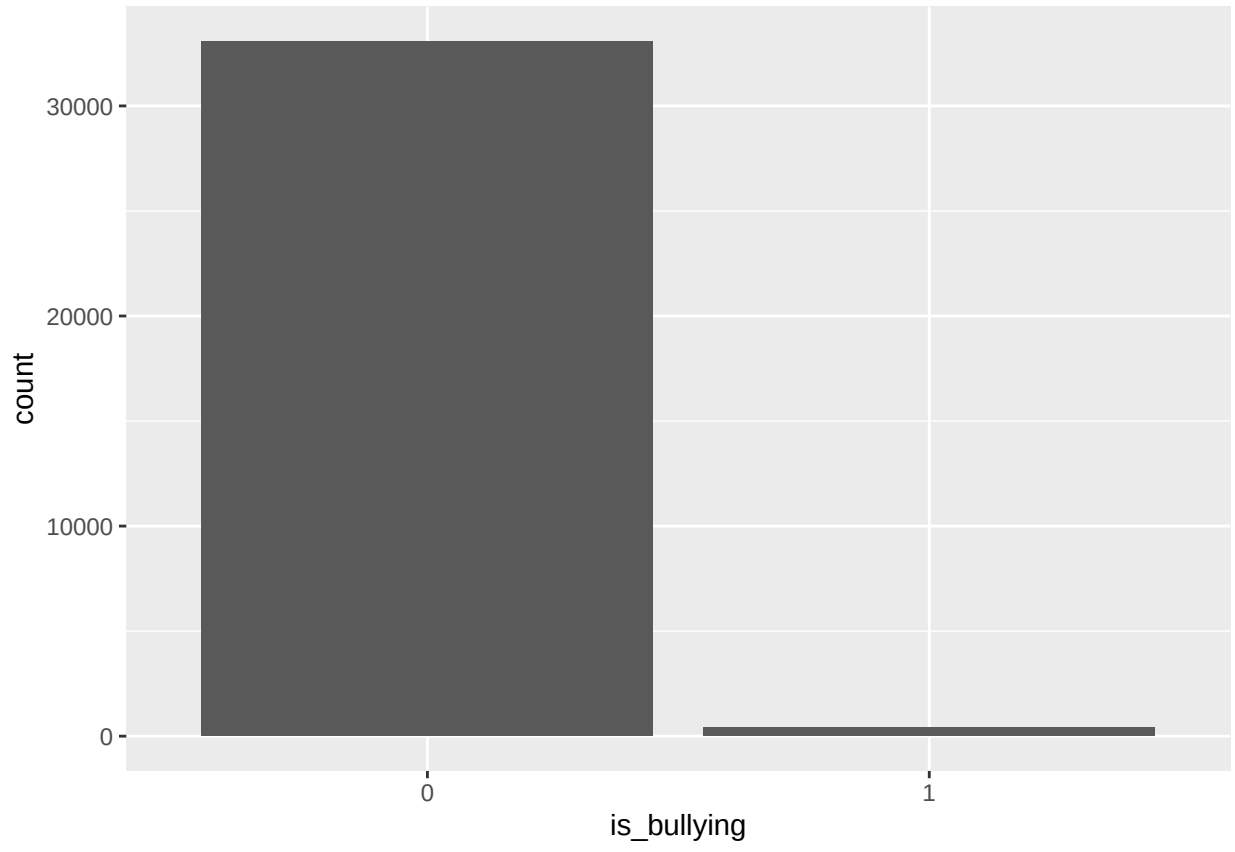
This code exports the clean posts to a csv file to be analysed separately. This also comes in handy in order to save time when performing EDAs as pre-processing can take time (especially on slow computers).

```
corpus = VCorpus(VectorSource(posts_df$html_message))
dtm = DocumentTermMatrix(corpus)
dtm = removeSparseTerms(dtm, 0.999)
posts_data = as.data.frame(as.matrix(dtm))
posts_data$is_bullying = as.factor(posts_df$is_bullying)
```

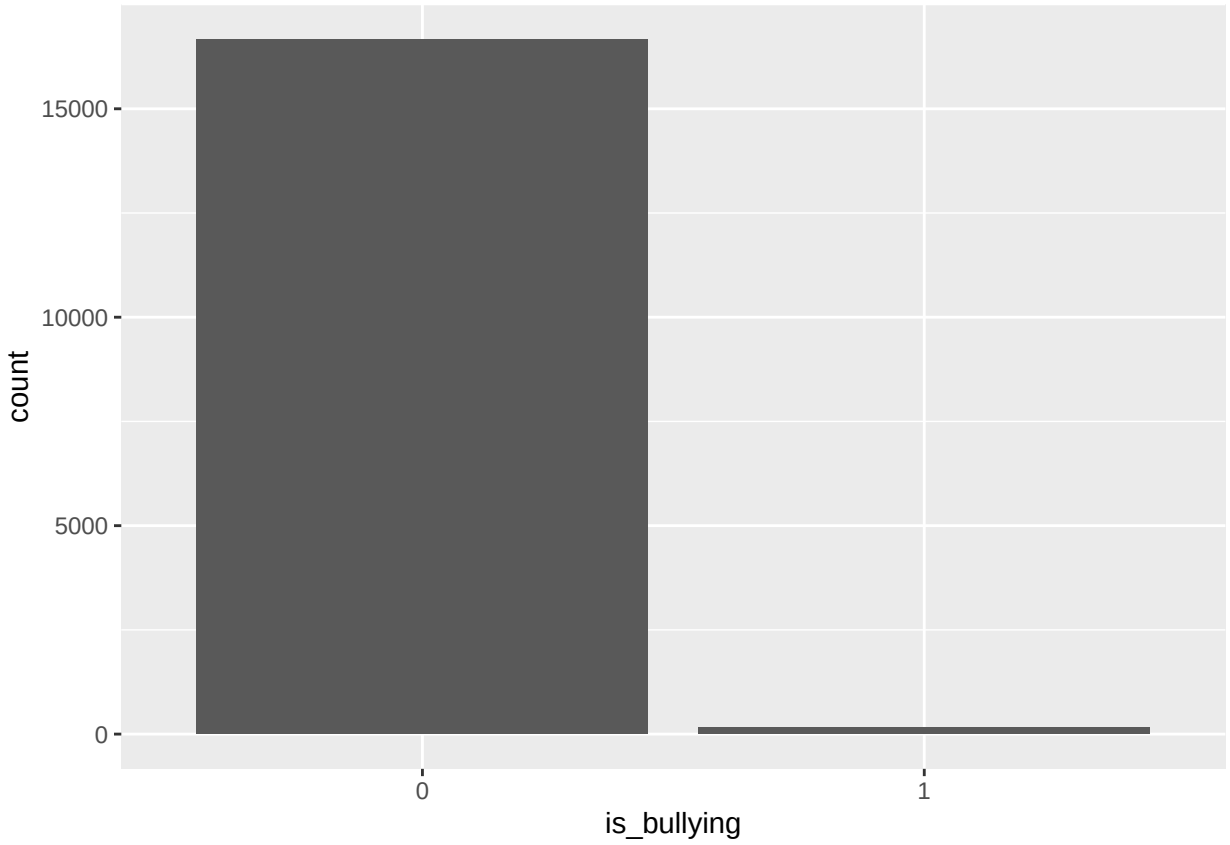
```
lol_posts <- posts_df %>% filter(dataset=="LoL")
wow_posts <- posts_df %>% filter(dataset=="WoW")

lol_corpus = VCorpus(VectorSource(lol_posts$html_message))
lol_dtm = DocumentTermMatrix(lol_corpus)
lol_dtm = removeSparseTerms(lol_dtm, 0.999)
lol_posts_data = as.data.frame(as.matrix(lol_dtm))
lol_posts_data$is_bullying = as.factor(lol_posts$is_bullying)

wow_corpus = VCorpus(VectorSource(wow_posts$html_message))
wow_dtm = DocumentTermMatrix(wow_corpus)
wow_dtm = removeSparseTerms(wow_dtm, 0.999)
wow_posts_data = as.data.frame(as.matrix(wow_dtm))
wow_posts_data$is_bullying = as.factor(wow_posts$is_bullying)
```

We create a document term matrix from the html messages and we remove sparse terms (i.e. empty values) using removeSparseTerms. We then assign a "is_bullying" column for model building and EDA purposes.

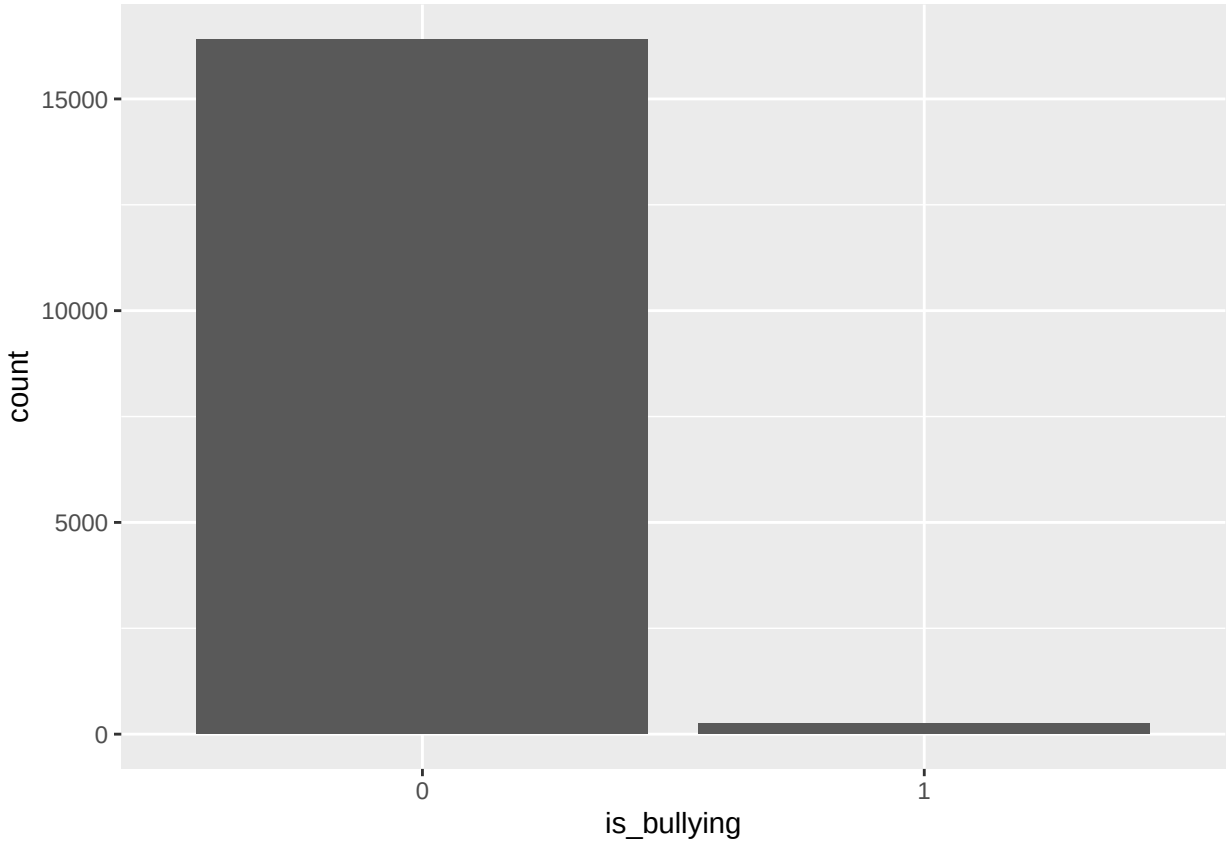
```
ggplot(data=posts_data, aes(x=is_bullying)) + geom_bar()
```



```
ggplot(data=wow_posts_data, aes(x=is_bullying)) + geom_bar()
```




```
ggplot(data=lol_posts_data, aes(x=is_bullying)) + geom_bar()
```



As we can see, the data is heavily imbalanced where there isn't many bullying cases. This will result in the classifier being trained to where it is more accurate at classifying non-bullying cases rather than bullying cases. We will use undersampling because we have plenty of non-bullying data but not enough data for bullying cases (this means we can afford to reduce how much data we are dealing with).

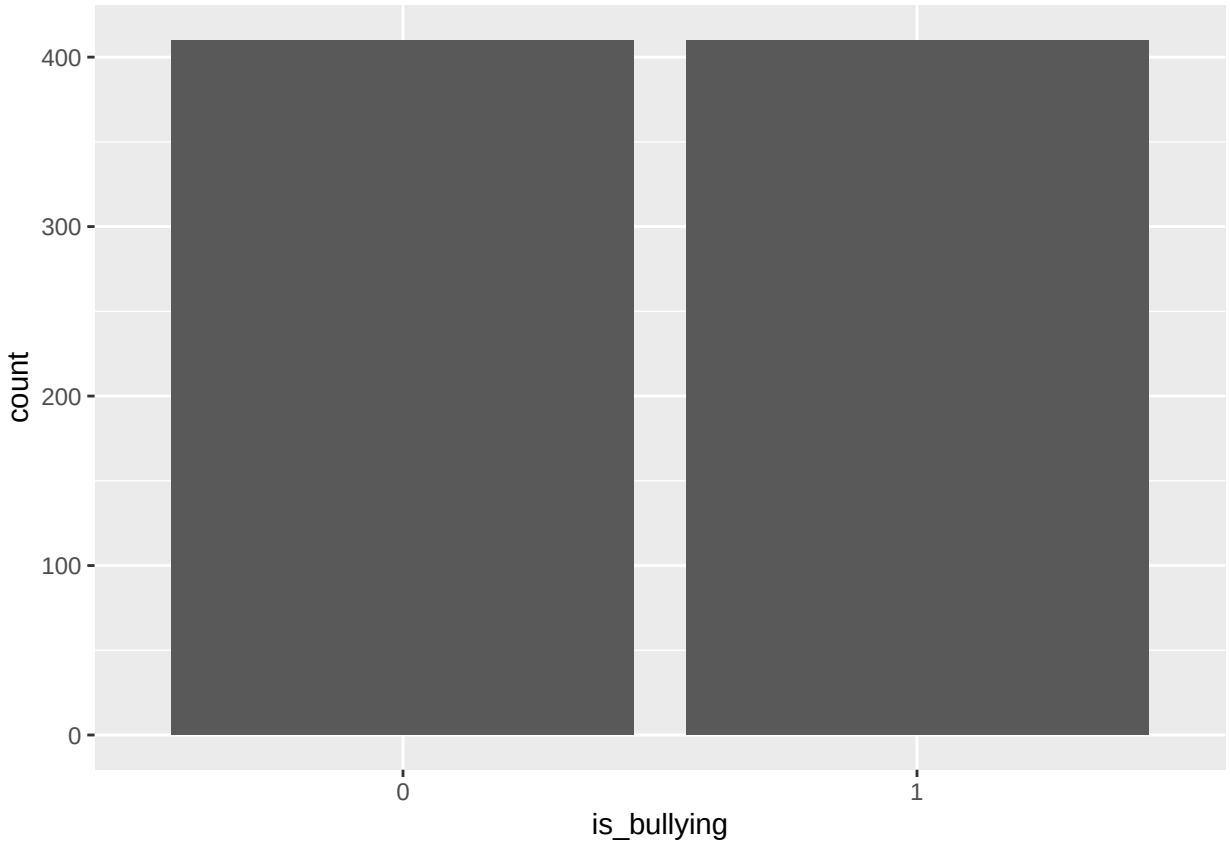
```
is_bullying = which(lol_posts_data$is_bullying == 1)
not_bullying = which(lol_posts_data$is_bullying == 0)
nsamp = min(length(is_bullying), length(not_bullying))
sample_bullying = sample(is_bullying, nsamp)
sample_not_bullying = sample(not_bullying, nsamp)
lol_posts_data_balanced = lol_posts_data[c(sample_bullying, sample_not_bullying),]
```

```
is_bullying = which(wow_posts_data$is_bullying == 1)
not_bullying = which(wow_posts_data$is_bullying == 0)
nsamp = min(length(is_bullying), length(not_bullying))
sample_bullying = sample(is_bullying, nsamp)
sample_not_bullying = sample(not_bullying, nsamp)
wow_posts_data_balanced = wow_posts_data[c(sample_bullying, sample_not_bullying),]
```

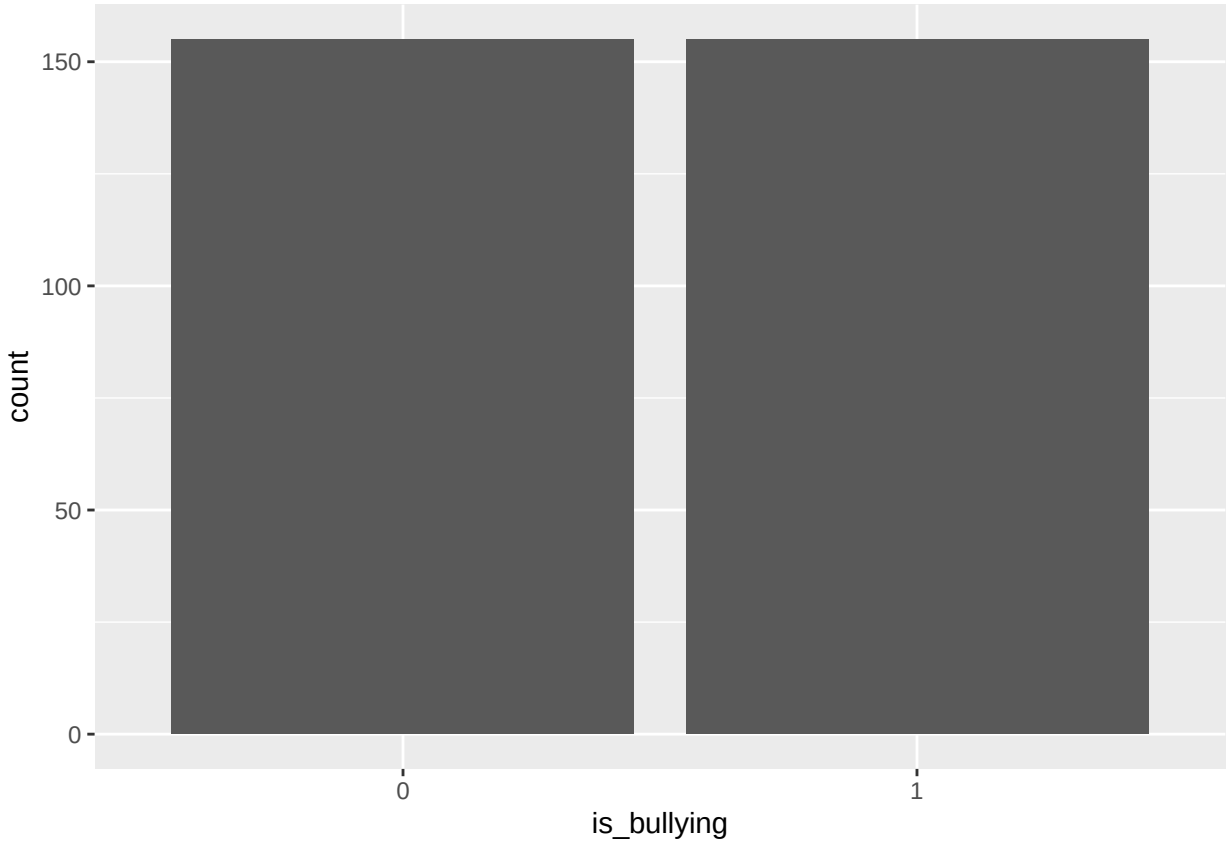
```
is_bullying = which(posts_data$is_bullying == 1)
not_bullying = which(posts_data$is_bullying == 0)
nsamp = min(length(is_bullying), length(not_bullying))
sample_bullying = sample(is_bullying, nsamp)
sample_not_bullying = sample(not_bullying, nsamp)
posts_data_balanced = posts_data[c(sample_bullying, sample_not_bullying),]
```

This creates a sample of the bullying data for balancing purposes. However, this comes at the expense of having much less data to work with as there are significantly less cyberbullying cases versus non-cyberbullying cases.

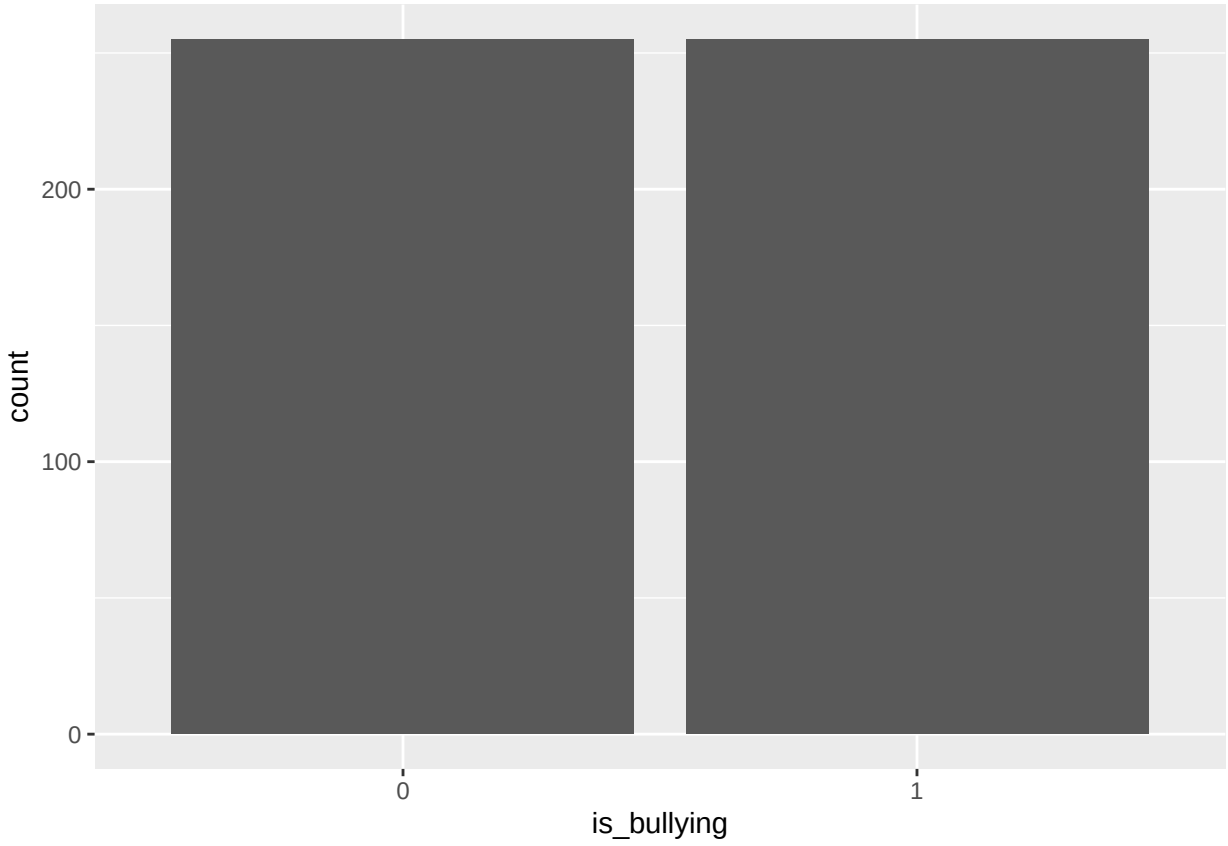
```
ggplot(data=posts_data_balanced, aes(x=is_bullying)) + geom_bar()
```




```
ggplot(data=wow_posts_data_balanced, aes(x=is_bullying)) + geom_bar()
```




```
ggplot(data=lol_posts_data_balanced, aes(x=is_bullying)) + geom_bar()
```




```
set.seed(42)
part <- sample(2, nrow(posts_data), replace=TRUE, prob=c(0.6, 0.4))
train <- posts_data[part == 1, ]
test <- posts_data[part == 2, ]
```

We split the data using a 60:40 train-test split.

```
set.seed(42)
part <- sample(2, nrow(posts_data_balanced), replace=TRUE, prob=c(0.6, 0.4))
train_balanced <- posts_data_balanced[part == 1, ]
test_balanced <- posts_data_balanced[part == 2, ]
```

We split the data using a 60:40 split. This is for the balanced data.

```
set.seed(42)
part <- sample(2, nrow(lol_posts_data_balanced), replace=TRUE, prob=c(0.6, 0.4))
lol_train_balanced <- lol_posts_data_balanced[part == 1, ]
lol_test_balanced <- lol_posts_data_balanced[part == 2, ]
```

```
set.seed(42)
part <- sample(2, nrow(wow_posts_data_balanced), replace=TRUE, prob=c(0.6, 0.4))
wow_train_balanced <- wow_posts_data_balanced[part == 1, ]
wow_test_balanced <- wow_posts_data_balanced[part == 2, ]
```

```
write.csv(posts_data_balanced, file="Data/clean_posts_dtm_balanced_sample.csv")
write.csv(wow_train_balanced, file="Data/clean_wow_train_balanced_sample.csv")
write.csv(wow_test_balanced, file="Data/clean_wow_test_balanced_sample.csv")
write.csv(lol_train_balanced, file="Data/clean_lol_train_balanced_sample.csv")
write.csv(lol_test_balanced, file="Data/clean_lol_test_balanced_sample.csv")
write.csv(train_balanced, file="Data/train_balanced.csv")
write.csv(test_balanced, file="Data/test_balanced.csv")
write.csv(train, file="Data/train.csv")
write.csv(test, file="Data/test.csv")
write.csv(posts_data, file="Data/clean_posts_dtm.csv")
```

We export the training and test data to make steps such as model building and EDA easier and more convenient as it can take time to pre-process the data (especially on slower computers).

Step 4: EDA

League of Legends analysis

```
posts_df %>% filter(dataset == "LoL") -> lol_posts
lol_posts %>% filter(is_bullying == 1) -> bullying_lol
bullying_lol
```

```
## # A tibble: 255 x 10
## # Groups:   id [255]
##    topic_id post_~1 author html_~2 times~3 dataset id    bully~4 is_bu~5 word_~6
##    <chr>      <int> <chr>  <chr>   <chr>   <chr>   <chr>   <dbl>   <int>   <int>
##  1 1030          11 160a1~ seen w~ 2014-1~ LoL     LoL_~       3       1      22
##  2 1030          12 1649d~ can te~ 2014-1~ LoL     LoL_~       2       1       9
##  3 1030          15 160a1~ fight ~ 2014-1~ LoL     LoL_~       3       1      42
##  4 1030          17 160a1~ dropha~ 2014-1~ LoL     LoL_~       2       1      25
##  5 1030          18 1649d~ get re~ 2014-1~ LoL     LoL_~       3       1      39
##  6 1030          19 160a1~ defend~ 2014-1~ LoL     LoL_~       3       1      30
##  7 1030          30 160a1~ still ~ 2014-1~ LoL     LoL_~       3       1       8
##  8 1030          33 d99c3~ uh not~ 2014-1~ LoL     LoL_~       3       1      13
##  9 1030          37 160a1~ refus ~ 2014-1~ LoL     LoL_~       3       1      18
## 10 1030          44 160a1~ reach ~ 2014-1~ LoL     LoL_~       3       1       7
## # ... with 245 more rows, and abbreviated variable names 1: post_number,
## #   2: html_message, 3: timestamp, 4: bullying_severity, 5: is_bullying,
## #   6: word_counts
```

```
nrow(bullying_lol) / nrow(lol_posts) * 100
```

```
## [1] 1.530429
```

For this data set 1.53% of the posts are labeled as bullying.

```
lol_tibble <- tibble(txt = lol_posts$html_message)
lol_tibble #transforming the HTML messages into a tibble for an easier workflow
```

```
## # A tibble: 16,662 x 1
##    txt
##    <chr>
##  1 bannd
##  2 can hope
##  3 fdcbbee dirti ing show
##  4 seen well coupl time irelia jarvan just havent seen mean happen vice versa c~
##  5 can tell ing kid much immatur ing littl gotcha
##  6 special snowflakeeeegif
##  7 gonna take bait fdcbbee drophack internet finish repli
##  8 fight fire fire good strategi also fair certain peopl agre talk admit popula~
##  9 kind peopl defend fdcbbee
## 10 drophack game season now get rest life lmao peopl disgust find someth better~
## # ... with 16,652 more rows
```

```
lol_tibble <- lol_tibble%>%
    mutate(linenumber = row_number()) %>%
 unnest_tokens(word, txt) %>% anti_join(stop_words)
lol_tibble #splitting tibble by words
```

```
## # A tibble: 334,514 x 2
##    linenumber word
##         <int> <chr>
##  1          1 bannd
##  2          2 hope
##  3          3 fdcbbee
##  4          3 dirti
##  5          3 ing
##  6          4 coupl
##  7          4 time
##  8          4 irelia
##  9          4 jarvan
## 10          4 havent
## # ... with 334,504 more rows
```

```
lol_counts <- lol_tibble %>% count(word, sort=TRUE)
lol_counts #sorting words by count
```

```
## # A tibble: 17,818 x 2
##    word         n
##    <chr>    <int>
##  1 lp        7020
##  2 ffsgive   6932
##  3 game      4991
##  4 play      3642
##  5 riot      3189
##  6 peopl     2998
##  7 time      2595
##  8 dominion  1978
##  9 realli    1913
## 10 player    1861
## # ... with 17,808 more rows
```

```
wordcloud(lol_counts$word, lol_counts$n, max.words = 250,
          min.freq=25, random.order=FALSE, colors=brewer.pal(8, "Dark2"))
```


```
  #creating a word cloud out of sorted list
```

By using a word cloud to analyse the HTML messages from the League of Legends boards, we have gained several insights into the issue of cyberbullying within the game.

The word helped us to identify the most common words or phrases used in the chat. This allowed us to see if certain language or terminology was frequently used in a negative or bullying context. For example, the words “stupid” and “idiot” were often marked as cyberbullying, which could indicate that players were using those words to mock or belittle others.

```
lol_bigrams <- tibble(txt = lol_posts$html_message) %>%
  unnest_tokens(bigram, txt, token = "ngrams", n = 2)
lol_bigrams
```

```
## # A tibble: 451,507 x 1
##    bigram
##    <chr>
##  1 <NA>
##  2 can hope
##  3 fdcbbee dirti
##  4 dirti ing
##  5 ing show
```

```
##  6 seen well
##  7 well coupl
##  8 coupl time
##  9 time irelia
## 10 irelia jarvan
## # ... with 451,497 more rows
```

```
#split original tibble into two-word bigrams
```

```
lol_bigrams <- lol_bigrams %>%
      separate(bigram, c("word1", "word2"), sep = " ")
lol_bigrams #separating them out for easier cleaning
```

```
## # A tibble: 451,507 x 2
##    word1   word2
##    <chr>   <chr>
##  1 <NA>    <NA>
##  2 can     hope
##  3 fdcbbee dirti
##  4 dirti   ing
##  5 ing     show
##  6 seen    well
##  7 well    coupl
##  8 coupl   time
##  9 time    irelia
## 10 irelia  jarvan
## # ... with 451,497 more rows
```

```
lol_bigrams <- lol_bigrams %>%
      filter(!word1 %in% stop_words$word) %>%
      filter(!word2 %in% stop_words$word)
lol_bigrams #filtering out unwanted words
```

```
## # A tibble: 231,542 x 2
##    word1   word2
##    <chr>   <chr>
##  1 <NA>    <NA>
##  2 fdcbbee dirti
##  3 dirti   ing
##  4 coupl   time
##  5 time    irelia
##  6 irelia  jarvan
##  7 happen  vice
##  8 vice    versa
##  9 assumpt level
## 10 level   intellig
## # ... with 231,532 more rows
```

```
lol_bigrams <- lol_bigrams %>%
            filter(!is.na(word1)) %>%
            filter(!is.na(word2))
lol_bigrams #removing null values
```

```
## # A tibble: 230,640 x 2
##    word1      word2
##    <chr>      <chr>
##  1 fdcbbee    dirti
##  2 dirti      ing
##  3 coupl      time
##  4 time       irelia
##  5 irelia     jarvan
##  6 happen     vice
##  7 vice       versa
##  8 assumpt    level
##  9 level      intellig
## 10 highschool retard
## # ... with 230,630 more rows
```

```
lol_bigrams <- lol_bigrams %>%
            unite(bigram, word1, word2, sep=" ")
lol_bigrams #joining the words back together
```

```
## # A tibble: 230,640 x 1
##    bigram
##    <chr>
##  1 fdcbbee dirti
##  2 dirti ing
##  3 coupl time
##  4 time irelia
##  5 irelia jarvan
##  6 happen vice
##  7 vice versa
##  8 assumpt level
##  9 level intellig
## 10 highschool retard
## # ... with 230,630 more rows
```

```
lol_bigram_counts <- lol_bigrams %>% count(bigram, sort=TRUE)
lol_bigram_counts #counting and sorting bigrams
```

```
## # A tibble: 154,477 x 2
##    bigram            n
##    <chr>         <int>
##  1 lp ffsgive     6932
##  2 late game       380
##  3 twin fang       347
##  4 play game       309
##  5 earli game      270
##  6 game mode       243
##  7 play dominion   209
##  8 rank queue      146
##  9 play rank       143
## 10 leagu legend    137
## # ... with 154,467 more rows
```

```
lol_bigram_counts %>%
  filter(str_detect(lol_bigram_counts$bigram,"[0-9]", negate = TRUE)) -> lol_bigram_counts
lol_bigram_counts #removing any bigrams with numbers
```

```
## # A tibble: 154,477 x 2
##    bigram            n
##    <chr>         <int>
##  1 lp ffsgive     6932
##  2 late game       380
##  3 twin fang       347
##  4 play game       309
##  5 earli game      270
##  6 game mode       243
##  7 play dominion   209
##  8 rank queue      146
##  9 play rank       143
## 10 leagu legend    137
## # ... with 154,467 more rows
```

```
lol_filtered_bigrams <- lol_bigram_counts %>%
                  filter(n >= 4)
lol_filtered_bigrams #sorting remaining bigrams with a frequency of 4 or more
```

```
## # A tibble: 6,031 x 2
##    bigram            n
##    <chr>         <int>
##  1 lp ffsgive     6932
##  2 late game       380
##  3 twin fang       347
##  4 play game       309
##  5 earli game      270
##  6 game mode       243
##  7 play dominion   209
##  8 rank queue      146
##  9 play rank       143
## 10 leagu legend    137
## # ... with 6,021 more rows
```

```
lol_separated_bigrams <- lol_filtered_bigrams %>%
  select("bigram") %>%
  separate(bigram, c("word1", "word2"), sep = " ")
lol_separated_bigrams #separating bigrams again, preparing for graphical representation
```

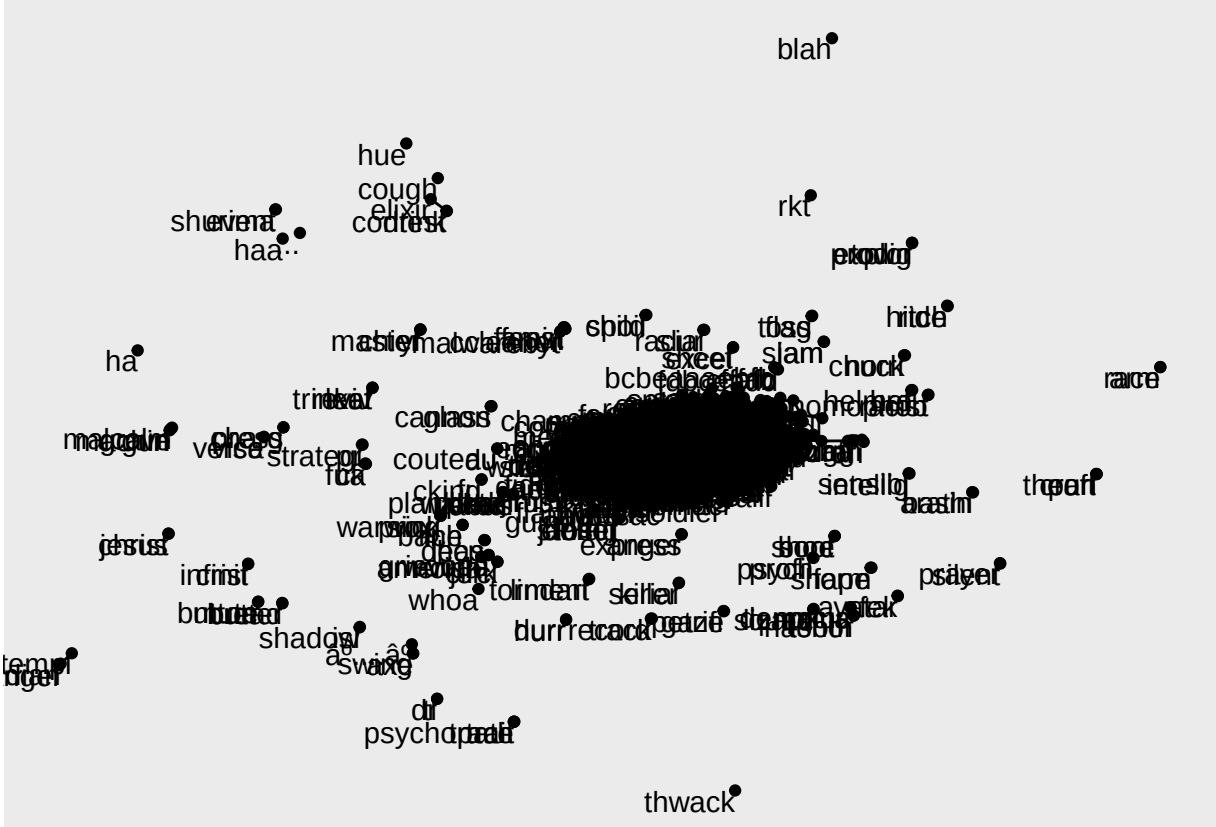
```
## # A tibble: 6,031 x 2
##    word1 word2
##    <chr> <chr>
##  1 lp    ffsgive
##  2 late  game
##  3 twin  fang
##  4 play  game
##  5 earli game
```

```
##  6 game  mode
##  7 play  dominion
##  8 rank  queue
##  9 play  rank
## 10 leagu legend
## # ... with 6,021 more rows
```

```
lol_bigram_graph <- lol_separated_bigrams %>%
                  graph_from_data_frame()
lol_bigram_graph  #creating bigram graph
```

```
## IGRAPH 0b1d9e0 DN-- 1761 6031 --
## + attr: name (v/c)
## + edges from 0b1d9e0 (vertex names):
##  [1] lp      ->ffsgive  late    ->game     twin    ->fang     play    ->game
##  [5] earli   ->game     game    ->mode     play    ->dominion rank    ->queue
##  [9] play    ->rank     leagu   ->legend   peopl   ->play     poison  ->mage
## [13] rank    ->dominion summon  ->rift     rank    ->game     dominion->player
## [17] player  ->base     game    ->play     share   ->account  account ->share
## [21] twist   ->treelin  lane    ->bulli    win     ->rate     loss    ->prevent
## [25] mana    ->cost     ap      ->ratio    enemi   ->team     lane    ->phase
## [29] everi   ->singl    fortun  ->teller   noxious ->blast    everi   ->time
## + ... omitted several edges
```

```
ggraph(lol_bigram_graph, layout = "fr") +
  geom_edge_link() +
  geom_node_point() +
  geom_node_text(aes(label = name), vjust = 1, hjust = 1)
```



For this project we used a bigram representation to quickly visualize the most common two-word phrases in this large text dataset. This helped us to get a sense of the collocation of words in the data, which gave important insights into the meaning and context of the text.

Additionally, using a bigram representation allowed us to easily identify any unusual or unexpected combinations of words that might be present in the dataset. This was particularly useful when working with unstructured data, as it allowed us to quickly see into some of the relationships between words within the text. By identifying the most common two-word phrases, we were able to see how words were collocating in the dataset, which gave us an idea of how they were being used and the relationships between them. This helped us to identify patterns and themes in the data that would have been difficult to discern by just looking at individual words.

In summary, using a bigram representation was a valuable tool for our exploratory data analysis, as it helped us to quickly gain a better understanding of our dataset by identifying patterns and themes, and also helped us to identify any unusual or unexpected combinations of words that might be present in the HTML message data.

World of Warcraft analysis

```
wow_posts <- posts_df %>% filter(dataset == "WoW")
posts_df %>% filter(is_bullying == 1) -> bullying_wow
bullying_wow
```

```
## # A tibble: 410 x 10
## # Groups:   id [410]
```

```
##    topic_id post_~1 author html_~2 times~3 dataset id    bully~4 is_bu~5 word_~6
##    <chr>      <int> <chr>  <chr>   <chr>   <chr>   <chr>   <dbl>   <int>   <int>
##  1 1030          11 160a1~ seen w~ 2014-1~ LoL     LoL_~       3       1      22
##  2 1030          12 1649d~ can te~ 2014-1~ LoL     LoL_~       2       1       9
##  3 1030          15 160a1~ fight ~ 2014-1~ LoL     LoL_~       3       1      42
##  4 1030          17 160a1~ dropha~ 2014-1~ LoL     LoL_~       2       1      25
##  5 1030          18 1649d~ get re~ 2014-1~ LoL     LoL_~       3       1      39
##  6 1030          19 160a1~ defend~ 2014-1~ LoL     LoL_~       3       1      30
##  7 1030          30 160a1~ still ~ 2014-1~ LoL     LoL_~       3       1       8
##  8 1030          33 d99c3~ uh not~ 2014-1~ LoL     LoL_~       3       1      13
##  9 1030          37 160a1~ refus ~ 2014-1~ LoL     LoL_~       3       1      18
## 10 1030          44 160a1~ reach ~ 2014-1~ LoL     LoL_~       3       1       7
## # ... with 400 more rows, and abbreviated variable names 1: post_number,
## #   2: html_message, 3: timestamp, 4: bullying_severity, 5: is_bullying,
## #   6: word_counts
```

```
nrow(bullying_wow) / nrow(wow_posts) * 100
```

```
## [1] 2.437285
```

For this data set 2.4% of the posts are labeled as bullying.

```
wow_tibble <- tibble(txt = wow_posts$html_message)
wow_tibble #transforming the HTML messages into a tibble for an easier workflow
```

```
## # A tibble: 16,822 x 1
##    txt
##    <chr>
##  1 snip flamebait remov ive never busi expans releas earlier instead feel like ~
##  2 peopl come forum express concern love game major part life someth chang make~
##  3 give impress cri instead provid construct feedback two differ thing
##  4 garrison alreadi feel like option timesink realli fun realli excit rather bo~
##  5 like expans enjoy gore brought still plenti thing feel pressur run daili man~
##  6 earth op get upvot great like game even us unhappi nice other find enjoy gam~
##  7 sub expir
##  8 spoken like true blizzard fanboy serious even play cata mop cata bad filthi ~
##  9 peopl taken forum complain remain silent enjoy sinc forev matter goodbad gam~
## 10 know pathet open anoth thread call peopl pathet give feedback oh give damn m~
## # ... with 16,812 more rows
```

```
wow_tibble <- wow_tibble%>%
    mutate(linenumber = row_number()) %>%
 unnest_tokens(word, txt) %>% anti_join(stop_words)
```

```
## Joining, by = "word"
```

```
wow_tibble #splitting tibble by words and removing stop words
```

```
## # A tibble: 404,475 x 2
##    linenumber word
##         <int> <chr>
```

```
##  1          1 snip
##  2          1 flamebait
##  3          1 remov
##  4          1 ive
##  5          1 busi
##  6          1 expans
##  7          1 releas
##  8          1 earlier
##  9          1 feel
## 10          1 log
## # ... with 404,465 more rows
```

```
wow_counts <- wow_tibble %>% count(word, sort=TRUE)
wow_counts #sorting and counting the remaining words
```

```
## # A tibble: 21,194 x 2
##    word         n
##    <chr>    <int>
##  1 peopl     6764
##  2 realm     6746
##  3 game      4584
##  4 server    4192
##  5 time      4165
##  6 connect   4138
##  7 play      3107
##  8 player    3107
##  9 blizzard  2893
## 10 realli    2312
## # ... with 21,184 more rows
```

```
wordcloud(wow_counts$word, wow_counts$n, max.words = 250,
          min.freq=25, random.order=FALSE, colors=brewer.pal(8, "Dark2"))
```

```
## Warning in wordcloud(wow_counts$word, wow_counts$n, max.words = 250, min.freq =
## 25, : complain could not be fit on page. It will not be plotted.
```

```
## Warning in wordcloud(wow_counts$word, wow_counts$n, max.words = 250, min.freq =
## 25, : anymor could not be fit on page. It will not be plotted.
```

```
## Warning in wordcloud(wow_counts$word, wow_counts$n, max.words = 250, min.freq =
## 25, : develop could not be fit on page. It will not be plotted.
```

```
## Warning in wordcloud(wow_counts$word, wow_counts$n, max.words = 250, min.freq =
## 25, : queue could not be fit on page. It will not be plotted.
```

```
## Warning in wordcloud(wow_counts$word, wow_counts$n, max.words = 250, min.freq =
## 25, : coupl could not be fit on page. It will not be plotted.
```

```
## Warning in wordcloud(wow_counts$word, wow_counts$n, max.words = 250, min.freq =
## 25, : silvermoon could not be fit on page. It will not be plotted.
```

```
## Warning in wordcloud(wow_counts$word, wow_counts$n, max.words = 250, min.freq =
## 25, : page could not be fit on page. It will not be plotted.

## Warning in wordcloud(wow_counts$word, wow_counts$n, max.words = 250, min.freq =
## 25, : suggest could not be fit on page. It will not be plotted.

## Warning in wordcloud(wow_counts$word, wow_counts$n, max.words = 250, min.freq =
## 25, : inform could not be fit on page. It will not be plotted.

## Warning in wordcloud(wow_counts$word, wow_counts$n, max.words = 250, min.freq =
## 25, : ignor could not be fit on page. It will not be plotted.

## Warning in wordcloud(wow_counts$word, wow_counts$n, max.words = 250, min.freq =
## 25, : spend could not be fit on page. It will not be plotted.

## Warning in wordcloud(wow_counts$word, wow_counts$n, max.words = 250, min.freq =
## 25, : medium could not be fit on page. It will not be plotted.

## Warning in wordcloud(wow_counts$word, wow_counts$n, max.words = 250, min.freq =
## 25, : follow could not be fit on page. It will not be plotted.

## Warning in wordcloud(wow_counts$word, wow_counts$n, max.words = 250, min.freq =
## 25, : launch could not be fit on page. It will not be plotted.

## Warning in wordcloud(wow_counts$word, wow_counts$n, max.words = 250, min.freq =
## 25, : stay could not be fit on page. It will not be plotted.

## Warning in wordcloud(wow_counts$word, wow_counts$n, max.words = 250, min.freq =
## 25, : onlin could not be fit on page. It will not be plotted.

## Warning in wordcloud(wow_counts$word, wow_counts$n, max.words = 250, min.freq =
## 25, : view could not be fit on page. It will not be plotted.

## Warning in wordcloud(wow_counts$word, wow_counts$n, max.words = 250, min.freq =
## 25, : decid could not be fit on page. It will not be plotted.
```


```r
#creating word cloud
```

We used a word cloud to quickly visualise the most common words in our large text dataset. This helped us to get a sense of the overall topic and context of the data, as well as identify any patterns or themes that might be present.

Additionally, using a word cloud allowed us to easily identify any outliers or unusual words that might be present in the dataset, which could then be further investigated. This was particularly useful when working with unstructured data, as it allowed us to quickly gain insights without having to manually sift through all of the text. Using a word cloud was a valuable tool in our exploratory data analysis and helped us to quickly gain a better understanding of the dataset.

```r
wow_bigrams <- tibble(txt = wow_posts$html_message) %>%
  unnest_tokens(bigram, txt, token = "ngrams", n = 2)
wow_bigrams #split original tibble into two-word bigrams
```

```
## # A tibble: 551,023 x 1
##    bigram
##    <chr>
##  1 snip flamebait
##  2 flamebait remov
##  3 remov ive
##  4 ive never
##  5 never busi
##  6 busi expans
```

```
##  7 expans releas
##  8 releas earlier
##  9 earlier instead
## 10 instead feel
## # ... with 551,013 more rows
```

```
wow_bigrams <- wow_bigrams %>%
      separate(bigram, c("word1", "word2"), sep = " ")
wow_bigrams #separating them out for easier cleaning
```

```
## # A tibble: 551,023 x 2
##    word1     word2
##    <chr>     <chr>
##  1 snip      flamebait
##  2 flamebait remov
##  3 remov     ive
##  4 ive       never
##  5 never     busi
##  6 busi      expans
##  7 expans    releas
##  8 releas    earlier
##  9 earlier   instead
## 10 instead   feel
## # ... with 551,013 more rows
```

```
wow_bigrams <- wow_bigrams %>%
      filter(!word1 %in% stop_words$word) %>%
      filter(!word2 %in% stop_words$word)
wow_bigrams #filtering out unwanted words
```

```
## # A tibble: 282,808 x 2
##    word1     word2
##    <chr>     <chr>
##  1 snip      flamebait
##  2 flamebait remov
##  3 remov     ive
##  4 busi      expans
##  5 expans    releas
##  6 releas    earlier
##  7 log       bore
##  8 bore      daili
##  9 daili     quest
## 10 creat     gear
## # ... with 282,798 more rows
```

```
wow_bigrams <- wow_bigrams %>%
            filter(!is.na(word1)) %>%
            filter(!is.na(word2))
wow_bigrams #removing null values
```

```
## # A tibble: 282,464 x 2
##    word1     word2
```

```
##    <chr>     <chr>
##  1 snip      flamebait
##  2 flamebait remov
##  3 remov     ive
##  4 busi      expans
##  5 expans    releas
##  6 releas    earlier
##  7 log       bore
##  8 bore      daili
##  9 daili     quest
## 10 creat     gear
## # ... with 282,454 more rows
```

```
wow_bigrams <- wow_bigrams %>%
           unite(bigram, word1, word2, sep=" ")
wow_bigrams #joining the words back together
```

```
## # A tibble: 282,464 x 1
##    bigram
##    <chr>
##  1 snip flamebait
##  2 flamebait remov
##  3 remov ive
##  4 busi expans
##  5 expans releas
##  6 releas earlier
##  7 log bore
##  8 bore daili
##  9 daili quest
## 10 creat gear
## # ... with 282,454 more rows
```

```
wow_bigram_counts <- wow_bigrams %>% count(bigram, sort=TRUE)
wow_bigram_counts #counting and sorting bigrams
```

```
## # A tibble: 183,782 x 2
##    bigram             n
##    <chr>          <int>
##  1 connect realm    599
##  2 realm connect    392
##  3 pvp realm        341
##  4 play game        334
##  5 pve realm        270
##  6 mani peopl       255
##  7 classic server   253
##  8 vanilla server   234
##  9 rp realm         222
## 10 lot peopl        211
## # ... with 183,772 more rows
```

```
wow_bigram_counts %>%
  filter(str_detect(wow_bigram_counts$bigram,"[0-9]", negate = TRUE)) -> wow_bigram_counts
wow_bigram_counts #removing any bigrams with numbers
```

```
## # A tibble: 183,782 x 2
##    bigram             n
##    <chr>          <int>
##  1 connect realm    599
##  2 realm connect    392
##  3 pvp realm        341
##  4 play game        334
##  5 pve realm        270
##  6 mani peopl       255
##  7 classic server   253
##  8 vanilla server   234
##  9 rp realm         222
## 10 lot peopl        211
## # ... with 183,772 more rows
```

```
wow_filtered_bigrams <- wow_bigram_counts %>%
                  filter(n >= 4)
wow_filtered_bigrams #sorting remaining bigrams with a frequency of 4 or more
```

```
## # A tibble: 8,398 x 2
##    bigram             n
##    <chr>          <int>
##  1 connect realm    599
##  2 realm connect    392
##  3 pvp realm        341
##  4 play game        334
##  5 pve realm        270
##  6 mani peopl       255
##  7 classic server   253
##  8 vanilla server   234
##  9 rp realm         222
## 10 lot peopl        211
## # ... with 8,388 more rows
```

```
wow_separated_bigrams <- wow_filtered_bigrams %>%
  select("bigram") %>%
  separate(bigram, c("word1", "word2"), sep = " ")
wow_separated_bigrams #separating bigrams again, preparing for graphical representation
```

```
## # A tibble: 8,398 x 2
##    word1   word2
##    <chr>   <chr>
##  1 connect realm
##  2 realm   connect
##  3 pvp     realm
##  4 play    game
##  5 pve     realm
##  6 mani    peopl
##  7 classic server
##  8 vanilla server
##  9 rp      realm
## 10 lot     peopl
## # ... with 8,388 more rows
```

```
wow_bigram_graph <- wow_separated_bigrams %>%
                  graph_from_data_frame()
wow_bigram_graph #creating bigram graph
```

```
## IGRAPH 18cf305 DN-- 1898 8398 --
## + attr: name (v/c)
## + edges from 18cf305 (vertex names):
##  [1] connect->realm       realm  ->connect     pvp    ->realm
##  [4] play   ->game        pve    ->realm       mani   ->peopl
##  [7] classic->server      vanilla->server      rp     ->realm
## [10] lot    ->peopl       classic->realm       fli    ->mount
## [13] popul  ->realm       peopl  ->play        argent ->dawn
## [16] hellfir->hellfir     low    ->popul       confirm->kill
## [19] play   ->wow         vanilla->realm       max    ->level
## [22] real   ->life        world  ->warcraft    defia  ->brotherhood
## + ... omitted several edges
```

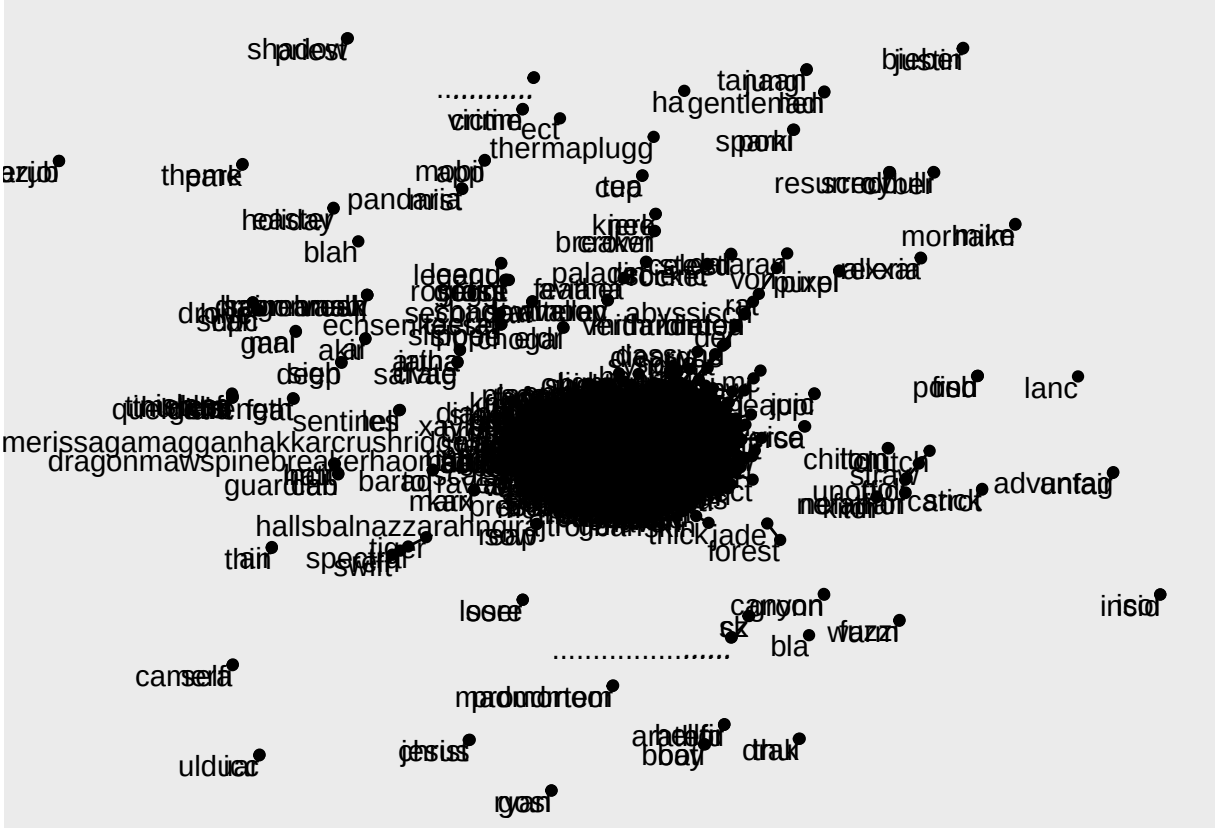
```
ggraph(wow_bigram_graph, layout = "fr") +
  geom_edge_link() +
  geom_node_point() +
  geom_node_text(aes(label = name), vjust = 1, hjust = 1)
```

```
## Warning: Using the `size` aesthetic in this geom was deprecated in ggplot2 3.4.0.
## i Please use `linewidth` in the `default_aes` field and elsewhere instead.

## Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on '      ' in 'mbcsToSbcs': dot substituted for <d1>

## Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on '      ' in 'mbcsToSbcs': dot substituted for <81>

## Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on '      ' in 'mbcsToSbcs': dot substituted for <d0>

## Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on '      ' in 'mbcsToSbcs': dot substituted for <b2>

## Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on '      ' in 'mbcsToSbcs': dot substituted for <d0>

## Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on '      ' in 'mbcsToSbcs': dot substituted for <b5>

## Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on '      ' in 'mbcsToSbcs': dot substituted for <d0>

## Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on '      ' in 'mbcsToSbcs': dot substituted for <b6>

## Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on '      ' in 'mbcsToSbcs': dot substituted for <d0>
```

```
## Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on '      ' in 'mbcsToSbcs': dot substituted for <b5>

## Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on '      ' in 'mbcsToSbcs': dot substituted for <d0>

## Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on '      ' in 'mbcsToSbcs': dot substituted for <b2>

## Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on '      ' in 'mbcsToSbcs': dot substituted for <d0>

## Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on '      ' in 'mbcsToSbcs': dot substituted for <b0>

## Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on '      ' in 'mbcsToSbcs': dot substituted for <d1>

## Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on '      ' in 'mbcsToSbcs': dot substituted for <82>

## Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on '      ' in 'mbcsToSbcs': dot substituted for <d0>

## Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on '      ' in 'mbcsToSbcs': dot substituted for <b5>

## Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on '      ' in 'mbcsToSbcs': dot substituted for <d0>

## Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on '      ' in 'mbcsToSbcs': dot substituted for <bb>

## Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on '      ' in 'mbcsToSbcs': dot substituted for <d1>

## Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on '      ' in 'mbcsToSbcs': dot substituted for <8c>

## Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on '    ' in 'mbcsToSbcs': dot substituted for <d0>

## Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on '    ' in 'mbcsToSbcs': dot substituted for <b2>

## Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on '    ' in 'mbcsToSbcs': dot substituted for <d0>

## Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on '    ' in 'mbcsToSbcs': dot substituted for <b5>
```

```
## Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on '   ' in 'mbcsToSbcs': dot substituted for <d1>

## Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on '   ' in 'mbcsToSbcs': dot substituted for <87>

## Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on '   ' in 'mbcsToSbcs': dot substituted for <d0>

## Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on '   ' in 'mbcsToSbcs': dot substituted for <bd>

## Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on '   ' in 'mbcsToSbcs': dot substituted for <d0>

## Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on '   ' in 'mbcsToSbcs': dot substituted for <b0>

## Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on '   ' in 'mbcsToSbcs': dot substituted for <d1>

## Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on '   ' in 'mbcsToSbcs': dot substituted for <8f>

## Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on '  ' in 'mbcsToSbcs': dot substituted for <d0>

## Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on '  ' in 'mbcsToSbcs': dot substituted for <b4>

## Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on '  ' in 'mbcsToSbcs': dot substituted for <d1>

## Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on '  ' in 'mbcsToSbcs': dot substituted for <83>

## Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on '  ' in 'mbcsToSbcs': dot substituted for <d1>

## Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on '  ' in 'mbcsToSbcs': dot substituted for <88>

## Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on '   ' in 'mbcsToSbcs': dot substituted for <d0>

## Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on '   ' in 'mbcsToSbcs': dot substituted for <bf>

## Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on '   ' in 'mbcsToSbcs': dot substituted for <d0>
```

```
## Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on '   ' in 'mbcsToSbcs': dot substituted for <b5>

## Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on '   ' in 'mbcsToSbcs': dot substituted for <d1>

## Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on '   ' in 'mbcsToSbcs': dot substituted for <81>

## Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on '   ' in 'mbcsToSbcs': dot substituted for <d0>

## Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on '   ' in 'mbcsToSbcs': dot substituted for <bd>

## Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on '   ' in 'mbcsToSbcs': dot substituted for <d1>

## Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on '   ' in 'mbcsToSbcs': dot substituted for <8f>
```



The bigram representation helped us to identify the most common two-word phrases used in the chat. This allowed us to see the distribution of words and understand how players were communicating with each other. For example, the phrase “commit suicide” can be seen in the bigram - messages with this phrase often were flagged as cyberbulling.

Step 5: Predictive modelling (combined dataset)

Initial setup

```
train_balanced$is_bullying = as.factor(train_balanced$is_bullying)
test_balanced$is_bullying = as.factor(test_balanced$is_bullying)

lol_train_balanced$is_bullying = as.factor(lol_train_balanced$is_bullying)
lol_test_balanced$is_bullying = as.factor(lol_test_balanced$is_bullying)
wow_train_balanced$is_bullying = as.factor(wow_train_balanced$is_bullying)
wow_test_balanced$is_bullying = as.factor(wow_test_balanced$is_bullying)

train_control = trainControl(method = "cv", number = 5)
```

SVM

```
set.seed(42)

tic()
svm_model = caret::train(is_bullying~., data=train_balanced , method ="svmLinear" , trControl = train_c
svm_toc <- toc(quiet=T)

tic()
svm_wow_model = caret::train(is_bullying~., data=wow_train_balanced, method ="svmLinear" , trControl =
svm_wow_toc <- toc(quiet=T)

tic()
svm_lol_model = caret::train(is_bullying~., data=lol_train_balanced, method ="svmLinear" , trControl =
svm_lol_toc <- toc(quiet=T)

svm_time_taken <- svm_toc$toc - svm_toc$tic
svm_lol_time_taken <- svm_lol_toc$toc - svm_lol_toc$tic
svm_wow_time_taken <- svm_wow_toc$toc - svm_wow_toc$tic

svm_pred_y = predict(svm_model, test_balanced)
svm_wow_pred_y = predict(svm_wow_model, wow_test_balanced)
svm_lol_pred_y = predict(svm_lol_model, lol_test_balanced)
```

Here we create the trained model, with “is_bullying” as an output as this is what we want to test The method is the type of model we want to use trControl takes control of our parameters

SVM (Support Vector Machine) is a model that uses classification algorithms for two-group classification problems. We used an SVM model here due to the fact our dataset is balanced. We also decided to try out a linear kernel as we believe that we are working with linearly separable datasets and we have found that linear kernels are common in text classification problems.

MLP

```r
set.seed(42)

tic()
mlp_model = caret::train(is_bullying~., data=train_balanced , method ="mlp" , trControl = train_control
mlp_toc <- toc(quiet=T)

tic()
mlp_wow_model = caret::train(is_bullying~., data=wow_train_balanced , method ="mlp" , trControl = train_
```

```
## Warning in snnsObject$setUnitName(num, iNames[[i]]): SNNS error message
## in setUnitName : SNNS-Kernel Error: Symbol pattern invalid (must match
## [A-Za-z][^|, ]*)

## Warning in snnsObject$setUnitName(num, iNames[[i]]): SNNS error message
## in setUnitName : SNNS-Kernel Error: Symbol pattern invalid (must match
## [A-Za-z][^|, ]*)

## Warning in snnsObject$setUnitName(num, iNames[[i]]): SNNS error message
## in setUnitName : SNNS-Kernel Error: Symbol pattern invalid (must match
## [A-Za-z][^|, ]*)

## Warning in snnsObject$setUnitName(num, iNames[[i]]): SNNS error message
## in setUnitName : SNNS-Kernel Error: Symbol pattern invalid (must match
## [A-Za-z][^|, ]*)

## Warning in snnsObject$setUnitName(num, iNames[[i]]): SNNS error message
## in setUnitName : SNNS-Kernel Error: Symbol pattern invalid (must match
## [A-Za-z][^|, ]*)

## Warning in snnsObject$setUnitName(num, iNames[[i]]): SNNS error message
## in setUnitName : SNNS-Kernel Error: Symbol pattern invalid (must match
## [A-Za-z][^|, ]*)

## Warning in snnsObject$setUnitName(num, iNames[[i]]): SNNS error message
## in setUnitName : SNNS-Kernel Error: Symbol pattern invalid (must match
## [A-Za-z][^|, ]*)

## Warning in snnsObject$setUnitName(num, iNames[[i]]): SNNS error message
## in setUnitName : SNNS-Kernel Error: Symbol pattern invalid (must match
## [A-Za-z][^|, ]*)

## Warning in snnsObject$setUnitName(num, iNames[[i]]): SNNS error message
## in setUnitName : SNNS-Kernel Error: Symbol pattern invalid (must match
## [A-Za-z][^|, ]*)

## Warning in snnsObject$setUnitName(num, iNames[[i]]): SNNS error message
## in setUnitName : SNNS-Kernel Error: Symbol pattern invalid (must match
## [A-Za-z][^|, ]*)

## Warning in snnsObject$setUnitName(num, iNames[[i]]): SNNS error message
## in setUnitName : SNNS-Kernel Error: Symbol pattern invalid (must match
## [A-Za-z][^|, ]*)
```

```
## Warning in snnsObject$setUnitName(num, iNames[[i]]): SNNS error message
## in setUnitName : SNNS-Kernel Error: Symbol pattern invalid (must match
## [A-Za-z][^|, ]*)

## Warning in snnsObject$setUnitName(num, iNames[[i]]): SNNS error message
## in setUnitName : SNNS-Kernel Error: Symbol pattern invalid (must match
## [A-Za-z][^|, ]*)

## Warning in snnsObject$setUnitName(num, iNames[[i]]): SNNS error message
## in setUnitName : SNNS-Kernel Error: Symbol pattern invalid (must match
## [A-Za-z][^|, ]*)

## Warning in snnsObject$setUnitName(num, iNames[[i]]): SNNS error message
## in setUnitName : SNNS-Kernel Error: Symbol pattern invalid (must match
## [A-Za-z][^|, ]*)

## Warning in snnsObject$setUnitName(num, iNames[[i]]): SNNS error message
## in setUnitName : SNNS-Kernel Error: Symbol pattern invalid (must match
## [A-Za-z][^|, ]*)
```

```
mlp_wow_toc <- toc(quiet=T)

tic()
mlp_lol_model = caret::train(is_bullying~., data=lol_train_balanced , method = "mlp" , trControl = trai
```

```
## Warning in snnsObject$setUnitName(num, iNames[[i]]): SNNS error message
## in setUnitName : SNNS-Kernel Error: Symbol pattern invalid (must match
## [A-Za-z][^|, ]*)

## Warning in snnsObject$setUnitName(num, iNames[[i]]): SNNS error message
## in setUnitName : SNNS-Kernel Error: Symbol pattern invalid (must match
## [A-Za-z][^|, ]*)

## Warning in snnsObject$setUnitName(num, iNames[[i]]): SNNS error message
## in setUnitName : SNNS-Kernel Error: Symbol pattern invalid (must match
## [A-Za-z][^|, ]*)

## Warning in snnsObject$setUnitName(num, iNames[[i]]): SNNS error message
## in setUnitName : SNNS-Kernel Error: Symbol pattern invalid (must match
## [A-Za-z][^|, ]*)

## Warning in snnsObject$setUnitName(num, iNames[[i]]): SNNS error message
## in setUnitName : SNNS-Kernel Error: Symbol pattern invalid (must match
## [A-Za-z][^|, ]*)

## Warning in snnsObject$setUnitName(num, iNames[[i]]): SNNS error message
## in setUnitName : SNNS-Kernel Error: Symbol pattern invalid (must match
## [A-Za-z][^|, ]*)

## Warning in snnsObject$setUnitName(num, iNames[[i]]): SNNS error message
## in setUnitName : SNNS-Kernel Error: Symbol pattern invalid (must match
## [A-Za-z][^|, ]*)

## Warning in snnsObject$setUnitName(num, iNames[[i]]): SNNS error message
```

```
## in setUnitName : SNNS-Kernel Error: Symbol pattern invalid (must match
## [A-Za-z][^|, ]*)

## Warning in snnsObject$setUnitName(num, iNames[[i]]): SNNS error message
## in setUnitName : SNNS-Kernel Error: Symbol pattern invalid (must match
## [A-Za-z][^|, ]*)

## Warning in snnsObject$setUnitName(num, iNames[[i]]): SNNS error message
## in setUnitName : SNNS-Kernel Error: Symbol pattern invalid (must match
## [A-Za-z][^|, ]*)

## Warning in snnsObject$setUnitName(num, iNames[[i]]): SNNS error message
## in setUnitName : SNNS-Kernel Error: Symbol pattern invalid (must match
## [A-Za-z][^|, ]*)

## Warning in snnsObject$setUnitName(num, iNames[[i]]): SNNS error message
## in setUnitName : SNNS-Kernel Error: Symbol pattern invalid (must match
## [A-Za-z][^|, ]*)

## Warning in snnsObject$setUnitName(num, iNames[[i]]): SNNS error message
## in setUnitName : SNNS-Kernel Error: Symbol pattern invalid (must match
## [A-Za-z][^|, ]*)

## Warning in snnsObject$setUnitName(num, iNames[[i]]): SNNS error message
## in setUnitName : SNNS-Kernel Error: Symbol pattern invalid (must match
## [A-Za-z][^|, ]*)

## Warning in snnsObject$setUnitName(num, iNames[[i]]): SNNS error message
## in setUnitName : SNNS-Kernel Error: Symbol pattern invalid (must match
## [A-Za-z][^|, ]*)

## Warning in snnsObject$setUnitName(num, iNames[[i]]): SNNS error message
## in setUnitName : SNNS-Kernel Error: Symbol pattern invalid (must match
## [A-Za-z][^|, ]*)

## Warning in snnsObject$setUnitName(num, iNames[[i]]): SNNS error message
## in setUnitName : SNNS-Kernel Error: Symbol pattern invalid (must match
## [A-Za-z][^|, ]*)

## Warning in snnsObject$setUnitName(num, iNames[[i]]): SNNS error message
## in setUnitName : SNNS-Kernel Error: Symbol pattern invalid (must match
## [A-Za-z][^|, ]*)

## Warning in snnsObject$setUnitName(num, iNames[[i]]): SNNS error message
## in setUnitName : SNNS-Kernel Error: Symbol pattern invalid (must match
## [A-Za-z][^|, ]*)

## Warning in snnsObject$setUnitName(num, iNames[[i]]): SNNS error message
## in setUnitName : SNNS-Kernel Error: Symbol pattern invalid (must match
## [A-Za-z][^|, ]*)

## Warning in snnsObject$setUnitName(num, iNames[[i]]): SNNS error message
## in setUnitName : SNNS-Kernel Error: Symbol pattern invalid (must match
## [A-Za-z][^|, ]*)
```

```
## Warning in snnsObject$setUnitName(num, iNames[[i]]): SNNS error message
## in setUnitName : SNNS-Kernel Error: Symbol pattern invalid (must match
## [A-Za-z][^|, ]*)

## Warning in snnsObject$setUnitName(num, iNames[[i]]): SNNS error message
## in setUnitName : SNNS-Kernel Error: Symbol pattern invalid (must match
## [A-Za-z][^|, ]*)

## Warning in snnsObject$setUnitName(num, iNames[[i]]): SNNS error message
## in setUnitName : SNNS-Kernel Error: Symbol pattern invalid (must match
## [A-Za-z][^|, ]*)

## Warning in snnsObject$setUnitName(num, iNames[[i]]): SNNS error message
## in setUnitName : SNNS-Kernel Error: Symbol pattern invalid (must match
## [A-Za-z][^|, ]*)

## Warning in snnsObject$setUnitName(num, iNames[[i]]): SNNS error message
## in setUnitName : SNNS-Kernel Error: Symbol pattern invalid (must match
## [A-Za-z][^|, ]*)

## Warning in snnsObject$setUnitName(num, iNames[[i]]): SNNS error message
## in setUnitName : SNNS-Kernel Error: Symbol pattern invalid (must match
## [A-Za-z][^|, ]*)

## Warning in snnsObject$setUnitName(num, iNames[[i]]): SNNS error message
## in setUnitName : SNNS-Kernel Error: Symbol pattern invalid (must match
## [A-Za-z][^|, ]*)

## Warning in snnsObject$setUnitName(num, iNames[[i]]): SNNS error message
## in setUnitName : SNNS-Kernel Error: Symbol pattern invalid (must match
## [A-Za-z][^|, ]*)

## Warning in snnsObject$setUnitName(num, iNames[[i]]): SNNS error message
## in setUnitName : SNNS-Kernel Error: Symbol pattern invalid (must match
## [A-Za-z][^|, ]*)

## Warning in snnsObject$setUnitName(num, iNames[[i]]): SNNS error message
## in setUnitName : SNNS-Kernel Error: Symbol pattern invalid (must match
## [A-Za-z][^|, ]*)

## Warning in snnsObject$setUnitName(num, iNames[[i]]): SNNS error message
## in setUnitName : SNNS-Kernel Error: Symbol pattern invalid (must match
## [A-Za-z][^|, ]*)
```

```
mlp_lol_toc <- toc(quiet=T)

mlp_time_taken <- mlp_toc$toc - mlp_toc$tic
mlp_wow_time_taken <- mlp_wow_toc$toc - mlp_wow_toc$tic
mlp_lol_time_taken <- mlp_lol_toc$toc - mlp_lol_toc$tic

mlp_pred_y = predict(mlp_model, test_balanced)
mlp_lol_pred_y = predict(mlp_lol_model, lol_test_balanced)
mlp_wow_pred_y = predict(mlp_wow_model, wow_test_balanced)
```

Here we create the trained model, with "is_bullying" as an output as this is what we want to test The method is the type of model we want to use trControl takes control of our parameters

MLP (Multilayer Perceptrons) are very flexible and can be used generally to learn a mapping from inputs to outputs. MLPs are suitable for classification prediction problems which is exactly what we want. They are comprised of one or more layers of neurons. Data is fed to the input layer, there may be one or more hidden layers providing levels of abstraction, and predictions are made on the output layer, also called the visible layer.

glmnet

```
set.seed(42)

tic()
glmnet_model = caret::train(is_bullying~., data=train_balanced , method = "glmnet" , trControl = train_
glmnet_toc <- toc(quiet=T)

tic()
glmnet_wow_model = caret::train(is_bullying~., data=wow_train_balanced , method = "glmnet" , trControl =
glmnet_wow_toc <- toc(quiet=T)

tic()
glmnet_lol_model = caret::train(is_bullying~., data=lol_train_balanced , method = "glmnet" , trControl =
glmnet_lol_toc <- toc(quiet=T)

glmnet_time_taken <- glmnet_toc$toc - glmnet_toc$tic
glmnet_wow_time_taken <- glmnet_wow_toc$toc - glmnet_wow_toc$tic
glmnet_lol_time_taken <- glmnet_lol_toc$toc - glmnet_lol_toc$tic

glmnet_pred_y = predict(glmnet_model, test_balanced)
glmnet_lol_pred_y = predict(glmnet_lol_model, lol_test_balanced)
glmnet_wow_pred_y = predict(glmnet_wow_model, wow_test_balanced)
```

We have utilized the glmnet package as it has extremely efficient procedures for utilising lasso or elastic-net regularization for logistic regression. Since this is a classification problem, logistic regression will be used.

When we use caret to train a model by the "glmnet" tag, caret will select a generalized linear model (logistic regression in this case) via penalized maximum likelihood. # Step 6: Hyperparameter Tuning

```
random_train_control_grid = caret::trainControl(method="cv", number=3, search="random")
```

We decided to go with a random grid search as it simplified the process of hyperparameter tuning as it removed any potential biases that may occur from an uneven distribution of knowledge for the different models trained (i.e. we thought that a random grid search would be much more consistent when comparing tuned models).

SVM Linear kernel

```
set.seed(42)
```

```
tic()
svm_linear_tuned = caret::train(is_bullying~ ., data = train_balanced, method = "svmLinear", trControl =
svm_linear_tuned_toc = toc(quiet=T)

tic()
svm_linear_wow_tuned = caret::train(is_bullying~ ., data = wow_train_balanced, method = "svmLinear", tr
svm_linear_wow_tuned_toc = toc(quiet=T)

tic()
svm_linear_lol_tuned = caret::train(is_bullying~ ., data = lol_train_balanced, method = "svmLinear", tr
svm_linear_lol_tuned_toc = toc(quiet=T)

svm_linear_tuned_time_taken <- svm_linear_tuned_toc$toc - svm_linear_tuned_toc$tic
svm_linear_wow_tuned_time_taken <- svm_linear_wow_tuned_toc$toc - svm_linear_wow_tuned_toc$tic
svm_linear_lol_tuned_time_taken <- svm_linear_lol_tuned_toc$toc - svm_linear_lol_tuned_toc$tic

svm_tuned_pred_y = predict(svm_linear_tuned, test_balanced)
svm_tuned_wow_pred_y = predict(svm_linear_wow_tuned, wow_test_balanced)
svm_tuned_lol_pred_y = predict(svm_linear_lol_tuned, lol_test_balanced)
```

SVM Polynomial kernel

```
set.seed(42)

tic()
svm_model_poly = caret::train(is_bullying~ ., data = train_balanced, method = "svmPoly", trControl = ra
svm_poly_toc = toc(quiet=T)

tic()
svm_lol_model_poly = caret::train(is_bullying~ ., data = lol_train_balanced, method = "svmPoly", trCont
svm_poly_lol_toc = toc(quiet=T)

tic()
svm_wow_model_poly = caret::train(is_bullying~ ., data = wow_train_balanced, method = "svmPoly", trCont
svm_poly_wow_toc = toc(quiet=T)

svm_poly_tuned_time_taken = svm_poly_toc$toc - svm_poly_toc$tic
svm_lol_poly_tuned_time_taken = svm_poly_lol_toc$toc - svm_poly_lol_toc$tic
svm_wow_poly_tuned_time_taken = svm_poly_wow_toc$toc - svm_poly_wow_toc$tic

svm_poly_pred_y = predict(svm_model_poly, test_balanced)
svm_lol_poly_pred_y = predict(svm_lol_model_poly, lol_test_balanced)
svm_wow_poly_pred_y = predict(svm_wow_model_poly, wow_test_balanced)
```

SVM RBF Kernel

```
set.seed(42)

tic()
```

```
svm_model_rbf = caret::train(is_bullying~ ., data = train_balanced, method = "svmRadial", trControl = ra
svm_rbf_toc = toc(quiet=T)

tic()
svm_lol_model_rbf = caret::train(is_bullying~ ., data = lol_train_balanced, method = "svmRadial", trCont
svm_lol_rbf_toc = toc(quiet=T)

tic()
svm_wow_model_rbf = caret::train(is_bullying~ ., data = wow_train_balanced, method = "svmRadial", trCont
svm_wow_rbf_toc = toc(quiet=T)

# summarizing the results
svm_rbf_tuned_time_taken = svm_rbf_toc$toc - svm_rbf_toc$tic
svm_wow_rbf_tuned_time_taken = svm_wow_rbf_toc$toc - svm_wow_rbf_toc$tic
svm_lol_rbf_tuned_time_taken = svm_lol_rbf_toc$toc - svm_lol_rbf_toc$tic

svm_rbf_pred_y = predict(svm_model_rbf, test_balanced)
svm_wow_rbf_pred_y = predict(svm_wow_model_rbf, wow_test_balanced)
svm_lol_rbf_pred_y = predict(svm_lol_model_rbf, lol_test_balanced)
```

MLP Tuned

```
set.seed(42)

tic()
mlp_model_tuned = caret::train(is_bullying~ ., data = train_balanced, method = "mlp", trControl = randor
mlp_toc = toc(quiet=T)

tic()
mlp_lol_model_tuned = caret::train(is_bullying~ ., data = lol_train_balanced, method = "mlp", trControl
```

```
## Warning in snnsObject$setUnitName(num, iNames[[i]]): SNNS error message
## in setUnitName : SNNS-Kernel Error: Symbol pattern invalid (must match
## [A-Za-z][^|, ]*)

## Warning in snnsObject$setUnitName(num, iNames[[i]]): SNNS error message
## in setUnitName : SNNS-Kernel Error: Symbol pattern invalid (must match
## [A-Za-z][^|, ]*)

## Warning in snnsObject$setUnitName(num, iNames[[i]]): SNNS error message
## in setUnitName : SNNS-Kernel Error: Symbol pattern invalid (must match
## [A-Za-z][^|, ]*)

## Warning in snnsObject$setUnitName(num, iNames[[i]]): SNNS error message
## in setUnitName : SNNS-Kernel Error: Symbol pattern invalid (must match
## [A-Za-z][^|, ]*)

## Warning in snnsObject$setUnitName(num, iNames[[i]]): SNNS error message
## in setUnitName : SNNS-Kernel Error: Symbol pattern invalid (must match
## [A-Za-z][^|, ]*)
```

```
## Warning in snnsObject$setUnitName(num, iNames[[i]]): SNNS error message
## in setUnitName : SNNS-Kernel Error: Symbol pattern invalid (must match
## [A-Za-z][^|, ]*)

## Warning in snnsObject$setUnitName(num, iNames[[i]]): SNNS error message
## in setUnitName : SNNS-Kernel Error: Symbol pattern invalid (must match
## [A-Za-z][^|, ]*)

## Warning in snnsObject$setUnitName(num, iNames[[i]]): SNNS error message
## in setUnitName : SNNS-Kernel Error: Symbol pattern invalid (must match
## [A-Za-z][^|, ]*)

## Warning in snnsObject$setUnitName(num, iNames[[i]]): SNNS error message
## in setUnitName : SNNS-Kernel Error: Symbol pattern invalid (must match
## [A-Za-z][^|, ]*)

## Warning in snnsObject$setUnitName(num, iNames[[i]]): SNNS error message
## in setUnitName : SNNS-Kernel Error: Symbol pattern invalid (must match
## [A-Za-z][^|, ]*)

## Warning in snnsObject$setUnitName(num, iNames[[i]]): SNNS error message
## in setUnitName : SNNS-Kernel Error: Symbol pattern invalid (must match
## [A-Za-z][^|, ]*)

## Warning in snnsObject$setUnitName(num, iNames[[i]]): SNNS error message
## in setUnitName : SNNS-Kernel Error: Symbol pattern invalid (must match
## [A-Za-z][^|, ]*)

## Warning in snnsObject$setUnitName(num, iNames[[i]]): SNNS error message
## in setUnitName : SNNS-Kernel Error: Symbol pattern invalid (must match
## [A-Za-z][^|, ]*)

## Warning in snnsObject$setUnitName(num, iNames[[i]]): SNNS error message
## in setUnitName : SNNS-Kernel Error: Symbol pattern invalid (must match
## [A-Za-z][^|, ]*)

## Warning in snnsObject$setUnitName(num, iNames[[i]]): SNNS error message
## in setUnitName : SNNS-Kernel Error: Symbol pattern invalid (must match
## [A-Za-z][^|, ]*)

## Warning in snnsObject$setUnitName(num, iNames[[i]]): SNNS error message
## in setUnitName : SNNS-Kernel Error: Symbol pattern invalid (must match
## [A-Za-z][^|, ]*)

## Warning in snnsObject$setUnitName(num, iNames[[i]]): SNNS error message
## in setUnitName : SNNS-Kernel Error: Symbol pattern invalid (must match
## [A-Za-z][^|, ]*)

## Warning in snnsObject$setUnitName(num, iNames[[i]]): SNNS error message
## in setUnitName : SNNS-Kernel Error: Symbol pattern invalid (must match
## [A-Za-z][^|, ]*)

## Warning in snnsObject$setUnitName(num, iNames[[i]]): SNNS error message
## in setUnitName : SNNS-Kernel Error: Symbol pattern invalid (must match
```

```
## [A-Za-z][^|, ]*)

## Warning in snnsObject$setUnitName(num, iNames[[i]]): SNNS error message
## in setUnitName : SNNS-Kernel Error: Symbol pattern invalid (must match
## [A-Za-z][^|, ]*)

## Warning in snnsObject$setUnitName(num, iNames[[i]]): SNNS error message
## in setUnitName : SNNS-Kernel Error: Symbol pattern invalid (must match
## [A-Za-z][^|, ]*)

## Warning in snnsObject$setUnitName(num, iNames[[i]]): SNNS error message
## in setUnitName : SNNS-Kernel Error: Symbol pattern invalid (must match
## [A-Za-z][^|, ]*)

## Warning in snnsObject$setUnitName(num, iNames[[i]]): SNNS error message
## in setUnitName : SNNS-Kernel Error: Symbol pattern invalid (must match
## [A-Za-z][^|, ]*)

## Warning in snnsObject$setUnitName(num, iNames[[i]]): SNNS error message
## in setUnitName : SNNS-Kernel Error: Symbol pattern invalid (must match
## [A-Za-z][^|, ]*)

## Warning in snnsObject$setUnitName(num, iNames[[i]]): SNNS error message
## in setUnitName : SNNS-Kernel Error: Symbol pattern invalid (must match
## [A-Za-z][^|, ]*)

## Warning in snnsObject$setUnitName(num, iNames[[i]]): SNNS error message
## in setUnitName : SNNS-Kernel Error: Symbol pattern invalid (must match
## [A-Za-z][^|, ]*)

## Warning in snnsObject$setUnitName(num, iNames[[i]]): SNNS error message
## in setUnitName : SNNS-Kernel Error: Symbol pattern invalid (must match
## [A-Za-z][^|, ]*)

## Warning in snnsObject$setUnitName(num, iNames[[i]]): SNNS error message
## in setUnitName : SNNS-Kernel Error: Symbol pattern invalid (must match
## [A-Za-z][^|, ]*)

## Warning in snnsObject$setUnitName(num, iNames[[i]]): SNNS error message
## in setUnitName : SNNS-Kernel Error: Symbol pattern invalid (must match
## [A-Za-z][^|, ]*)

## Warning in snnsObject$setUnitName(num, iNames[[i]]): SNNS error message
## in setUnitName : SNNS-Kernel Error: Symbol pattern invalid (must match
## [A-Za-z][^|, ]*)

## Warning in snnsObject$setUnitName(num, iNames[[i]]): SNNS error message
## in setUnitName : SNNS-Kernel Error: Symbol pattern invalid (must match
## [A-Za-z][^|, ]*)

## Warning in snnsObject$setUnitName(num, iNames[[i]]): SNNS error message
## in setUnitName : SNNS-Kernel Error: Symbol pattern invalid (must match
## [A-Za-z][^|, ]*)
```

```
mlp_lol_toc = toc(quiet=T)

tic()
mlp_wow_model_tuned = caret::train(is_bullying~ ., data = wow_train_balanced, method = "mlp", trControl
```

```
## Warning in snnsObject$setUnitName(num, iNames[[i]]): SNNS error message
## in setUnitName : SNNS-Kernel Error: Symbol pattern invalid (must match
## [A-Za-z][^|, ]*)

## Warning in snnsObject$setUnitName(num, iNames[[i]]): SNNS error message
## in setUnitName : SNNS-Kernel Error: Symbol pattern invalid (must match
## [A-Za-z][^|, ]*)

## Warning in snnsObject$setUnitName(num, iNames[[i]]): SNNS error message
## in setUnitName : SNNS-Kernel Error: Symbol pattern invalid (must match
## [A-Za-z][^|, ]*)

## Warning in snnsObject$setUnitName(num, iNames[[i]]): SNNS error message
## in setUnitName : SNNS-Kernel Error: Symbol pattern invalid (must match
## [A-Za-z][^|, ]*)

## Warning in snnsObject$setUnitName(num, iNames[[i]]): SNNS error message
## in setUnitName : SNNS-Kernel Error: Symbol pattern invalid (must match
## [A-Za-z][^|, ]*)

## Warning in snnsObject$setUnitName(num, iNames[[i]]): SNNS error message
## in setUnitName : SNNS-Kernel Error: Symbol pattern invalid (must match
## [A-Za-z][^|, ]*)

## Warning in snnsObject$setUnitName(num, iNames[[i]]): SNNS error message
## in setUnitName : SNNS-Kernel Error: Symbol pattern invalid (must match
## [A-Za-z][^|, ]*)

## Warning in snnsObject$setUnitName(num, iNames[[i]]): SNNS error message
## in setUnitName : SNNS-Kernel Error: Symbol pattern invalid (must match
## [A-Za-z][^|, ]*)

## Warning in snnsObject$setUnitName(num, iNames[[i]]): SNNS error message
## in setUnitName : SNNS-Kernel Error: Symbol pattern invalid (must match
## [A-Za-z][^|, ]*)

## Warning in snnsObject$setUnitName(num, iNames[[i]]): SNNS error message
## in setUnitName : SNNS-Kernel Error: Symbol pattern invalid (must match
## [A-Za-z][^|, ]*)
```

```
mlp_wow_toc = toc(quiet=T)

mlp_tuned_time_taken = mlp_toc$toc - mlp_toc$tic
mlp_lol_tuned_time_taken = mlp_lol_toc$toc - mlp_lol_toc$tic
mlp_wow_tuned_time_taken = mlp_wow_toc$toc - mlp_wow_toc$tic

mlp_tuned_pred_y = predict(mlp_model_tuned, test_balanced)
mlp_lol_tuned_pred_y = predict(mlp_lol_model_tuned, lol_test_balanced)
```

```
mlp_wow_tuned_pred_y = predict(mlp_wow_model_tuned, wow_test_balanced)
```

GLMNET Tuned

```
set.seed(42)

tic()
glm_tuned_model = caret::train(is_bullying~., data=train_balanced , method = "glmnet" , trControl = ran
glm_tuned_toc <- toc(quiet=T)

tic()
glm_lol_tuned_model = caret::train(is_bullying~., data=lol_train_balanced , method = "glmnet" , trContr
glm_lol_tuned_toc <- toc(quiet=T)

tic()
glm_wow_tuned_model = caret::train(is_bullying~., data=wow_train_balanced , method = "glmnet" , trContr
glm_wow_tuned_toc <- toc(quiet=T)

glm_tuned_time_taken <- glm_tuned_toc$toc - glm_tuned_toc$tic
glm_lol_tuned_time_taken <- glm_lol_tuned_toc$toc - glm_lol_tuned_toc$tic
glm_wow_tuned_time_taken <- glm_wow_tuned_toc$toc - glm_wow_tuned_toc$tic

glm_tuned_pred_y = predict(glm_tuned_model, test_balanced)
glm_lol_tuned_pred_y = predict(glm_lol_tuned_model, lol_test_balanced)
glm_wow_tuned_pred_y = predict(glm_wow_tuned_model, wow_test_balanced)
```

Step 7: Evaluation

For this evaluation, we wanted a comprehensive evaluation of the models involved using key metrics such as recall, precision, f1 score, the time taken to train the models and more to give a better idea as to which models perform best. However, we think that the F1 score is the most important metric as we believe that a cyberbullying classifier should minimize the number of false positives (i.e. people who haven't actually bullied anybody) and false negatives (i.e. bullies that haven't been identified). The F1 score acts as a trade-off between precision and recall which is exactly what we need. The F1 score is also robust against uneven class distributions unlike accuracy which gives us flexibility in how we should tweak pre-processing.

We have utilised the following to create an effective evaluation:

1. Created confusion matrices that illustrate where misclassifications are taking place
2. Plotted ROC curves to display the model performance
3. Created a table of metrics showing the recall, precision, f1 score and the time taken to train the models We wanted to show how long it took to train each model as this would help identify production costs for each model (the models would take much longer to train on big batches of data).

Creating confusion matrices (combined dataset)

```
svm_confusion_matrix <- caret::confusionMatrix(svm_pred_y, test_balanced$is_bullying)
mlp_confusion_matrix <- caret::confusionMatrix(mlp_pred_y, test_balanced$is_bullying)
glm_confusion_matrix <- caret::confusionMatrix(glmnet_pred_y, test_balanced$is_bullying)
```

```
svm_confusion_matrix
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 124  51
##          1  28 107
##
##                Accuracy : 0.7452
##                  95% CI : (0.6928, 0.7927)
##     No Information Rate : 0.5097
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.4916
##
##  Mcnemar's Test P-Value : 0.01332
##
##             Sensitivity : 0.8158
##             Specificity : 0.6772
##          Pos Pred Value : 0.7086
##          Neg Pred Value : 0.7926
##              Prevalence : 0.4903
##          Detection Rate : 0.4000
##    Detection Prevalence : 0.5645
##       Balanced Accuracy : 0.7465
##
##        'Positive' Class : 0
##
```

```
mlp_confusion_matrix
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 111  41
##          1  41 117
##
##                Accuracy : 0.7355
##                  95% CI : (0.6827, 0.7837)
##     No Information Rate : 0.5097
##     P-Value [Acc > NIR] : 3.742e-16
##
##                   Kappa : 0.4708
##
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 0.7303
##             Specificity : 0.7405
##          Pos Pred Value : 0.7303
##          Neg Pred Value : 0.7405
##              Prevalence : 0.4903
```

```
##          Detection Rate : 0.3581
##    Detection Prevalence : 0.4903
##       Balanced Accuracy : 0.7354
##
##        'Positive' Class : 0
##
```

```
glm_confusion_matrix
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 125  43
##          1  27 115
##
##                Accuracy : 0.7742
##                  95% CI : (0.7235, 0.8195)
##     No Information Rate : 0.5097
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.5491
##
##  Mcnemar's Test P-Value : 0.073
##
##             Sensitivity : 0.8224
##             Specificity : 0.7278
##          Pos Pred Value : 0.7440
##          Neg Pred Value : 0.8099
##              Prevalence : 0.4903
##          Detection Rate : 0.4032
##    Detection Prevalence : 0.5419
##       Balanced Accuracy : 0.7751
##
##        'Positive' Class : 0
##
```

When examining these confusion matrices, it would appear that the number of false positives (i.e. falsely classifying a post as cyberbullying) is the differentiating factor in terms of f1-score as the glmnet model has the least amount of false positives and it has the greatest f1-score (as we can see in the table of metrics below).

```
svm_linear_tuned_cf_matrix <- caret::confusionMatrix(svm_tuned_pred_y, test_balanced$is_bullying)
svm_poly_cf_matrix <- caret::confusionMatrix(svm_poly_pred_y, test_balanced$is_bullying)
svm_rbf_cf_matrix <- caret::confusionMatrix(svm_rbf_pred_y, test_balanced$is_bullying)

mlp_tuned_cf_matrix <- caret::confusionMatrix(mlp_tuned_pred_y, test_balanced$is_bullying)
glm_tuned_cf_matrix <- caret::confusionMatrix(glm_tuned_pred_y, test_balanced$is_bullying)
```

Here we create confusion matrices to get values such as F1 score, precision and recalls.

```
svm_linear_tuned_cf_matrix
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 122  52
##          1  30 106
##
##                Accuracy : 0.7355
##                  95% CI : (0.6827, 0.7837)
##     No Information Rate : 0.5097
##     P-Value [Acc > NIR] : 3.742e-16
##
##                   Kappa : 0.4722
##
##  Mcnemar's Test P-Value : 0.02039
##
##             Sensitivity : 0.8026
##             Specificity : 0.6709
##          Pos Pred Value : 0.7011
##          Neg Pred Value : 0.7794
##              Prevalence : 0.4903
##          Detection Rate : 0.3935
##    Detection Prevalence : 0.5613
##       Balanced Accuracy : 0.7368
##
##        'Positive' Class : 0
##
```

```
svm_poly_cf_matrix
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 124  56
##          1  28 102
##
##                Accuracy : 0.729
##                  95% CI : (0.6759, 0.7777)
##     No Information Rate : 0.5097
##     P-Value [Acc > NIR] : 2.621e-15
##
##                   Kappa : 0.4598
##
##  Mcnemar's Test P-Value : 0.00322
##
##             Sensitivity : 0.8158
##             Specificity : 0.6456
##          Pos Pred Value : 0.6889
##          Neg Pred Value : 0.7846
##              Prevalence : 0.4903
```

```
##          Detection Rate : 0.4000
##    Detection Prevalence : 0.5806
##       Balanced Accuracy : 0.7307
##
##        'Positive' Class : 0
##
```

```
svm_rbf_cf_matrix
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 127  45
##          1  25 113
##
##                Accuracy : 0.7742
##                  95% CI : (0.7235, 0.8195)
##     No Information Rate : 0.5097
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.5493
##
##  Mcnemar's Test P-Value : 0.02315
##
##             Sensitivity : 0.8355
##             Specificity : 0.7152
##          Pos Pred Value : 0.7384
##          Neg Pred Value : 0.8188
##              Prevalence : 0.4903
##          Detection Rate : 0.4097
##    Detection Prevalence : 0.5548
##       Balanced Accuracy : 0.7754
##
##        'Positive' Class : 0
##
```

```
mlp_tuned_cf_matrix
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 110  39
##          1  42 119
##
##                Accuracy : 0.7387
##                  95% CI : (0.6861, 0.7867)
##     No Information Rate : 0.5097
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.477
##
```

```
##  Mcnemar's Test P-Value : 0.8241
##
##             Sensitivity : 0.7237
##             Specificity : 0.7532
##          Pos Pred Value : 0.7383
##          Neg Pred Value : 0.7391
##              Prevalence : 0.4903
##          Detection Rate : 0.3548
##    Detection Prevalence : 0.4806
##       Balanced Accuracy : 0.7384
##
##        'Positive' Class : 0
##
```

It would appear that tuning MLP parameters through random grid search has actually decreased the accuracy of the model.

```
glm_tuned_cf_matrix
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 140  55
##          1  12 103
##
##                Accuracy : 0.7839
##                  95% CI : (0.7338, 0.8284)
##     No Information Rate : 0.5097
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.5699
##
##  Mcnemar's Test P-Value : 2.88e-07
##
##             Sensitivity : 0.9211
##             Specificity : 0.6519
##          Pos Pred Value : 0.7179
##          Neg Pred Value : 0.8957
##              Prevalence : 0.4903
##          Detection Rate : 0.4516
##    Detection Prevalence : 0.6290
##       Balanced Accuracy : 0.7865
##
##        'Positive' Class : 0
##
```

Creating confusion matrices (WoW dataset)

```
svm_wow_confusion_matrix <- caret::confusionMatrix(svm_wow_pred_y, wow_test_balanced$is_bullying)
mlp_wow_confusion_matrix <- caret::confusionMatrix(mlp_wow_pred_y, wow_test_balanced$is_bullying)
glm_wow_confusion_matrix <- caret::confusionMatrix(glmnet_wow_pred_y, wow_test_balanced$is_bullying)
```

```
svm_wow_confusion_matrix
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 36 28
##          1  9 47
##
##                Accuracy : 0.6917
##                  95% CI : (0.6009, 0.7727)
##     No Information Rate : 0.625
##     P-Value [Acc > NIR] : 0.077298
##
##                   Kappa : 0.3934
##
##  Mcnemar's Test P-Value : 0.003085
##
##             Sensitivity : 0.8000
##             Specificity : 0.6267
##          Pos Pred Value : 0.5625
##          Neg Pred Value : 0.8393
##              Prevalence : 0.3750
##          Detection Rate : 0.3000
##    Detection Prevalence : 0.5333
##       Balanced Accuracy : 0.7133
##
##        'Positive' Class : 0
##
```

```
mlp_wow_confusion_matrix
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 40 26
##          1  5 49
##
##                Accuracy : 0.7417
##                  95% CI : (0.6538, 0.8172)
##     No Information Rate : 0.625
##     P-Value [Acc > NIR] : 0.004627
##
##                   Kappa : 0.4959
##
##  Mcnemar's Test P-Value : 0.000328
##
##             Sensitivity : 0.8889
##             Specificity : 0.6533
##          Pos Pred Value : 0.6061
##          Neg Pred Value : 0.9074
##              Prevalence : 0.3750
```

```
##          Detection Rate : 0.3333
##    Detection Prevalence : 0.5500
##       Balanced Accuracy : 0.7711
##
##        'Positive' Class : 0
##
```

```
glm_wow_confusion_matrix
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 38 19
##          1  7 56
##
##                Accuracy : 0.7833
##                  95% CI : (0.6989, 0.8533)
##     No Information Rate : 0.625
##     P-Value [Acc > NIR] : 0.0001488
##
##                   Kappa : 0.5612
##
##  Mcnemar's Test P-Value : 0.0309841
##
##             Sensitivity : 0.8444
##             Specificity : 0.7467
##          Pos Pred Value : 0.6667
##          Neg Pred Value : 0.8889
##              Prevalence : 0.3750
##          Detection Rate : 0.3167
##    Detection Prevalence : 0.4750
##       Balanced Accuracy : 0.7956
##
##        'Positive' Class : 0
##
```

When examining these confusion matrices, it would appear that the number of false positives (i.e. falsely classifying a post as cyberbullying) is the differentiating factor in terms of f1-score as the glmnet model has the least amount of false positives and it has the greatest f1-score (as we can see in the table of metrics below).

```
svm_linear_tuned_cf_matrix <- caret::confusionMatrix(svm_tuned_wow_pred_y, wow_test_balanced$is_bullying
svm_poly_cf_matrix <- caret::confusionMatrix(svm_wow_poly_pred_y, wow_test_balanced$is_bullying)
svm_rbf_cf_matrix <- caret::confusionMatrix(svm_wow_rbf_pred_y, wow_test_balanced$is_bullying)

mlp_tuned_cf_matrix <- caret::confusionMatrix(mlp_wow_tuned_pred_y, wow_test_balanced$is_bullying)
glm_tuned_cf_matrix <- caret::confusionMatrix(glm_wow_tuned_pred_y, wow_test_balanced$is_bullying)
```

Here we create confusion matrices to get values such as F1 score, precision and recalls.

```
svm_linear_tuned_cf_matrix
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 37 27
##          1  8 48
##
##                Accuracy : 0.7083
##                  95% CI : (0.6184, 0.7877)
##     No Information Rate : 0.625
##     P-Value [Acc > NIR] : 0.034950
##
##                   Kappa : 0.4262
##
##  Mcnemar's Test P-Value : 0.002346
##
##             Sensitivity : 0.8222
##             Specificity : 0.6400
##          Pos Pred Value : 0.5781
##          Neg Pred Value : 0.8571
##              Prevalence : 0.3750
##          Detection Rate : 0.3083
##    Detection Prevalence : 0.5333
##       Balanced Accuracy : 0.7311
##
##        'Positive' Class : 0
##
```

```
svm_poly_cf_matrix
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 36 28
##          1  9 47
##
##                Accuracy : 0.6917
##                  95% CI : (0.6009, 0.7727)
##     No Information Rate : 0.625
##     P-Value [Acc > NIR] : 0.077298
##
##                   Kappa : 0.3934
##
##  Mcnemar's Test P-Value : 0.003085
##
##             Sensitivity : 0.8000
##             Specificity : 0.6267
##          Pos Pred Value : 0.5625
##          Neg Pred Value : 0.8393
##              Prevalence : 0.3750
```

```
##          Detection Rate : 0.3000
##    Detection Prevalence : 0.5333
##       Balanced Accuracy : 0.7133
##
##        'Positive' Class : 0
##
```

```
svm_rbf_cf_matrix
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 39 21
##          1  6 54
##
##                Accuracy : 0.775
##                  95% CI : (0.6898, 0.8462)
##     No Information Rate : 0.625
##     P-Value [Acc > NIR] : 0.000323
##
##                   Kappa : 0.55
##
##  Mcnemar's Test P-Value : 0.007054
##
##             Sensitivity : 0.8667
##             Specificity : 0.7200
##          Pos Pred Value : 0.6500
##          Neg Pred Value : 0.9000
##              Prevalence : 0.3750
##          Detection Rate : 0.3250
##    Detection Prevalence : 0.5000
##       Balanced Accuracy : 0.7933
##
##        'Positive' Class : 0
##
```

```
mlp_tuned_cf_matrix
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 39 23
##          1  6 52
##
##                Accuracy : 0.7583
##                  95% CI : (0.6717, 0.8318)
##     No Information Rate : 0.625
##     P-Value [Acc > NIR] : 0.001330
##
##                   Kappa : 0.5207
##
```

```
##  Mcnemar's Test P-Value : 0.002967
##
##             Sensitivity : 0.8667
##             Specificity : 0.6933
##          Pos Pred Value : 0.6290
##          Neg Pred Value : 0.8966
##              Prevalence : 0.3750
##          Detection Rate : 0.3250
##    Detection Prevalence : 0.5167
##       Balanced Accuracy : 0.7800
##
##        'Positive' Class : 0
##
```

It would appear that tuning MLP parameters through random grid search has actually decreased the accuracy of the model.

```
glm_tuned_cf_matrix
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 37 19
##          1  8 56
##
##                Accuracy : 0.775
##                  95% CI : (0.6898, 0.8462)
##     No Information Rate : 0.625
##     P-Value [Acc > NIR] : 0.000323
##
##                   Kappa : 0.5424
##
##  Mcnemar's Test P-Value : 0.054292
##
##             Sensitivity : 0.8222
##             Specificity : 0.7467
##          Pos Pred Value : 0.6607
##          Neg Pred Value : 0.8750
##              Prevalence : 0.3750
##          Detection Rate : 0.3083
##    Detection Prevalence : 0.4667
##       Balanced Accuracy : 0.7844
##
##        'Positive' Class : 0
##
```

Creating confusion matrices (LoL dataset)

```
svm_lol_confusion_matrix <- caret::confusionMatrix(svm_lol_pred_y, lol_test_balanced$is_bullying)
mlp_lol_confusion_matrix <- caret::confusionMatrix(mlp_lol_pred_y, lol_test_balanced$is_bullying)
glm_lol_confusion_matrix <- caret::confusionMatrix(glmnet_lol_pred_y, lol_test_balanced$is_bullying)
```

```
svm_lol_confusion_matrix
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 61 31
##          1 25 73
##
##                Accuracy : 0.7053
##                  95% CI : (0.6349, 0.7691)
##     No Information Rate : 0.5474
##     P-Value [Acc > NIR] : 6.002e-06
##
##                   Kappa : 0.4088
##
##  Mcnemar's Test P-Value : 0.504
##
##             Sensitivity : 0.7093
##             Specificity : 0.7019
##          Pos Pred Value : 0.6630
##          Neg Pred Value : 0.7449
##              Prevalence : 0.4526
##          Detection Rate : 0.3211
##    Detection Prevalence : 0.4842
##       Balanced Accuracy : 0.7056
##
##        'Positive' Class : 0
##
```

```
mlp_lol_confusion_matrix
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 67 41
##          1 19 63
##
##                Accuracy : 0.6842
##                  95% CI : (0.613, 0.7496)
##     No Information Rate : 0.5474
##     P-Value [Acc > NIR] : 8.178e-05
##
##                   Kappa : 0.3765
##
##  Mcnemar's Test P-Value : 0.006706
##
##             Sensitivity : 0.7791
##             Specificity : 0.6058
##          Pos Pred Value : 0.6204
##          Neg Pred Value : 0.7683
##              Prevalence : 0.4526
```

```
##          Detection Rate : 0.3526
##    Detection Prevalence : 0.5684
##       Balanced Accuracy : 0.6924
##
##        'Positive' Class : 0
##
```

```
glm_lol_confusion_matrix
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 75 36
##          1 11 68
##
##                Accuracy : 0.7526
##                  95% CI : (0.685, 0.8122)
##     No Information Rate : 0.5474
##     P-Value [Acc > NIR] : 3.931e-09
##
##                   Kappa : 0.513
##
##  Mcnemar's Test P-Value : 0.0004639
##
##             Sensitivity : 0.8721
##             Specificity : 0.6538
##          Pos Pred Value : 0.6757
##          Neg Pred Value : 0.8608
##              Prevalence : 0.4526
##          Detection Rate : 0.3947
##    Detection Prevalence : 0.5842
##       Balanced Accuracy : 0.7630
##
##        'Positive' Class : 0
##
```

When examining these confusion matrices, it would appear that the number of false positives (i.e. falsely classifying a post as cyberbullying) is the differentiating factor in terms of f1-score as the glmnet model has the least amount of false positives and it has the greatest f1-score (as we can see in the table of metrics below).

```
svm_lol_linear_tuned_cf_matrix <- caret::confusionMatrix(svm_tuned_lol_pred_y, lol_test_balanced$is_bul
svm_lol_poly_cf_matrix <- caret::confusionMatrix(svm_lol_poly_pred_y, lol_test_balanced$is_bullying)
svm_lol_rbf_cf_matrix <- caret::confusionMatrix(svm_lol_rbf_pred_y, lol_test_balanced$is_bullying)

mlp_lol_tuned_cf_matrix <- caret::confusionMatrix(mlp_lol_tuned_pred_y, lol_test_balanced$is_bullying)
glm_lol_tuned_cf_matrix <- caret::confusionMatrix(glm_lol_tuned_pred_y, lol_test_balanced$is_bullying)
```

Here we create confusion matrices to get values such as F1 score, precision and recalls.

```
svm_lol_linear_tuned_cf_matrix
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 76 54
##          1 10 50
##
##                Accuracy : 0.6632
##                  95% CI : (0.5912, 0.73)
##     No Information Rate : 0.5474
##     P-Value [Acc > NIR] : 0.0007693
##
##                   Kappa : 0.349
##
##  Mcnemar's Test P-Value : 7.658e-08
##
##             Sensitivity : 0.8837
##             Specificity : 0.4808
##          Pos Pred Value : 0.5846
##          Neg Pred Value : 0.8333
##              Prevalence : 0.4526
##          Detection Rate : 0.4000
##    Detection Prevalence : 0.6842
##       Balanced Accuracy : 0.6822
##
##        'Positive' Class : 0
##
```

```
svm_lol_poly_cf_matrix
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 79 71
##          1  7 33
##
##                Accuracy : 0.5895
##                  95% CI : (0.516, 0.6602)
##     No Information Rate : 0.5474
##     P-Value [Acc > NIR] : 0.137
##
##                   Kappa : 0.2216
##
##  Mcnemar's Test P-Value : 9.796e-13
##
##             Sensitivity : 0.9186
##             Specificity : 0.3173
##          Pos Pred Value : 0.5267
##          Neg Pred Value : 0.8250
##              Prevalence : 0.4526
```

```
##          Detection Rate : 0.4158
##    Detection Prevalence : 0.7895
##       Balanced Accuracy : 0.6180
##
##        'Positive' Class : 0
##
```

```
svm_lol_rbf_cf_matrix
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 43 23
##          1 43 81
##
##                Accuracy : 0.6526
##                  95% CI : (0.5803, 0.7201)
##     No Information Rate : 0.5474
##     P-Value [Acc > NIR] : 0.002065
##
##                   Kappa : 0.2846
##
##  Mcnemar's Test P-Value : 0.019349
##
##             Sensitivity : 0.5000
##             Specificity : 0.7788
##          Pos Pred Value : 0.6515
##          Neg Pred Value : 0.6532
##              Prevalence : 0.4526
##          Detection Rate : 0.2263
##    Detection Prevalence : 0.3474
##       Balanced Accuracy : 0.6394
##
##        'Positive' Class : 0
##
```

```
mlp_lol_tuned_cf_matrix
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 67 38
##          1 19 66
##
##                Accuracy : 0.7
##                  95% CI : (0.6294, 0.7642)
##     No Information Rate : 0.5474
##     P-Value [Acc > NIR] : 1.195e-05
##
##                   Kappa : 0.4059
##
```

```
##  Mcnemar's Test P-Value : 0.01712
##
##             Sensitivity : 0.7791
##             Specificity : 0.6346
##          Pos Pred Value : 0.6381
##          Neg Pred Value : 0.7765
##              Prevalence : 0.4526
##          Detection Rate : 0.3526
##    Detection Prevalence : 0.5526
##       Balanced Accuracy : 0.7068
##
##        'Positive' Class : 0
##
```

It would appear that tuning MLP parameters through random grid search has actually decreased the accuracy of the model.

```
glm_lol_tuned_cf_matrix
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 70 30
##          1 16 74
##
##                Accuracy : 0.7579
##                  95% CI : (0.6906, 0.817)
##     No Information Rate : 0.5474
##     P-Value [Acc > NIR] : 1.525e-09
##
##                   Kappa : 0.5182
##
##  Mcnemar's Test P-Value : 0.05527
##
##             Sensitivity : 0.8140
##             Specificity : 0.7115
##          Pos Pred Value : 0.7000
##          Neg Pred Value : 0.8222
##              Prevalence : 0.4526
##          Detection Rate : 0.3684
##    Detection Prevalence : 0.5263
##       Balanced Accuracy : 0.7627
##
##        'Positive' Class : 0
##
```

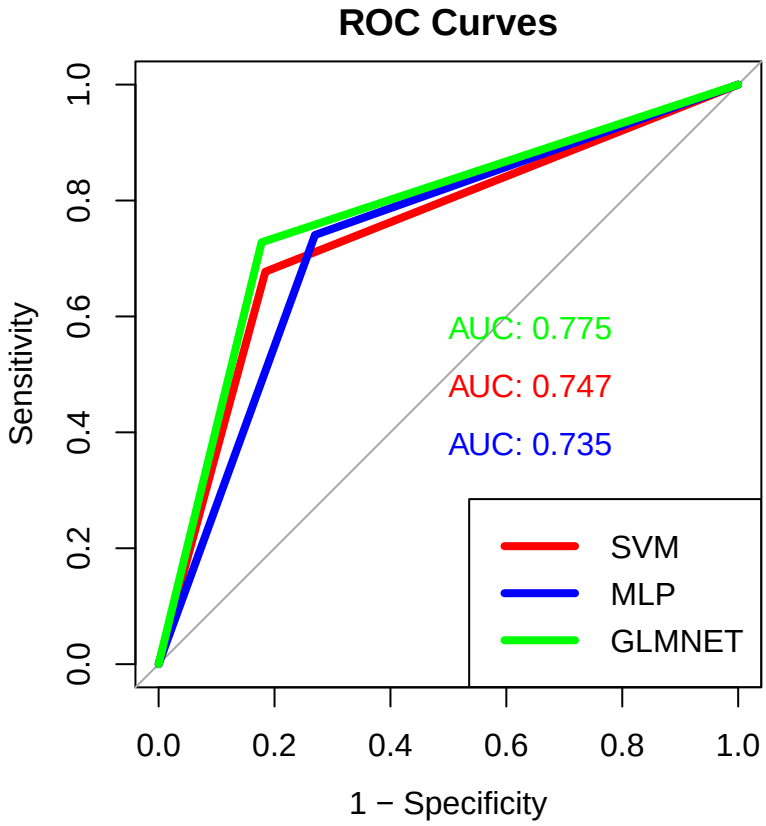
ROC Curves (combined dataset)

```
plot_roc_curves <- function(model_1_pred_y, model_2_pred_y, model_3_pred_y, model_names, test_data) {
  model_1_pred_y <- as.numeric(levels(model_1_pred_y))[model_1_pred_y]
```

```
  model_2_pred_y <- as.numeric(levels(model_2_pred_y))[model_2_pred_y]
  model_3_pred_y <- as.numeric(levels(model_3_pred_y))[model_3_pred_y]

  par(pty="s")
  model_1_roc <- roc(test_data$is_bullying~model_1_pred_y, plot=TRUE, print.auc=TRUE, col="red", lwd=4,
  model_2_roc <- roc(test_data$is_bullying~model_2_pred_y, plot=TRUE, print.auc=TRUE, print.auc.y=0.4,
  model_3_roc <- roc(test_data$is_bullying, model_3_pred_y, plot=TRUE, print.auc=TRUE, print.auc.y=0.6,

  legend("bottomright", legend=model_names, col=c("red", "blue", "green"), lwd=4)
}
```

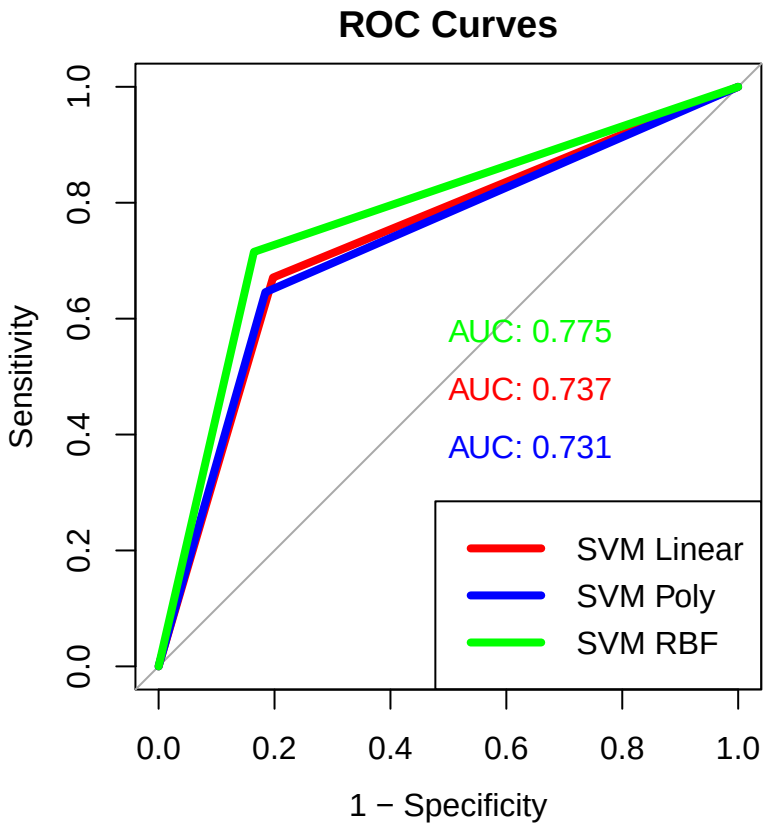
Non-tuned models

```
plot_roc_curves(svm_pred_y, mlp_pred_y, glmnet_pred_y, c("SVM", "MLP", "GLMNET"), test_balanced)
```



Comparing tuned SVM models

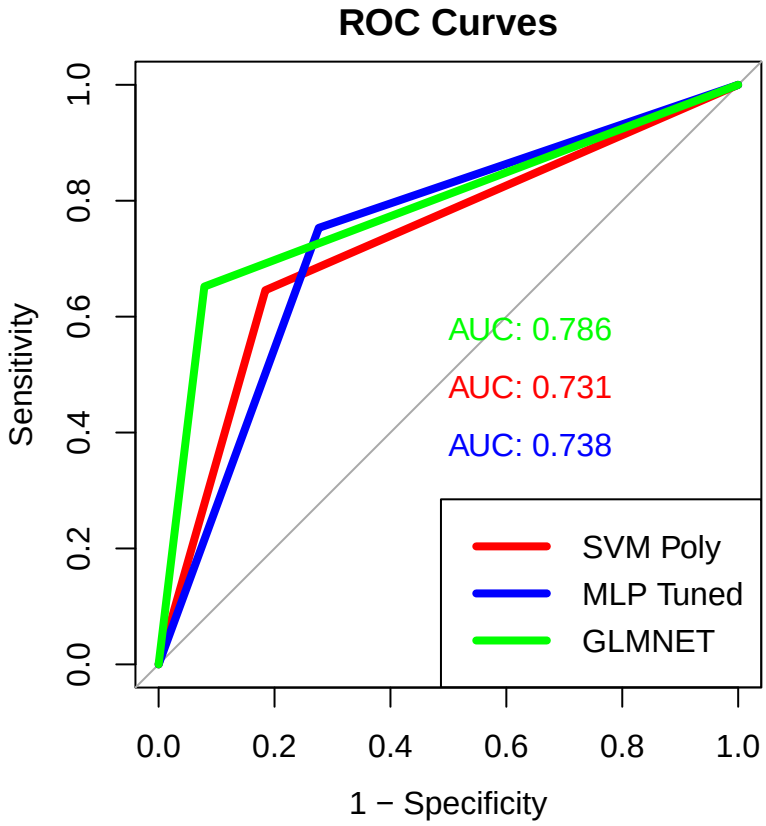
```
plot_roc_curves(svm_tuned_pred_y, svm_poly_pred_y, svm_rbf_pred_y, c("SVM Linear", "SVM Poly", "SVM RBF
```



It would appear that the polynomial kernel is most suitable for bullying classification. We will compare this SVM model that uses the polynomial kernel with other models.

Tuned models

```
plot_roc_curves(svm_poly_pred_y, mlp_tuned_pred_y, glm_tuned_pred_y, c("SVM Poly", "MLP Tuned", "GLMNET
```

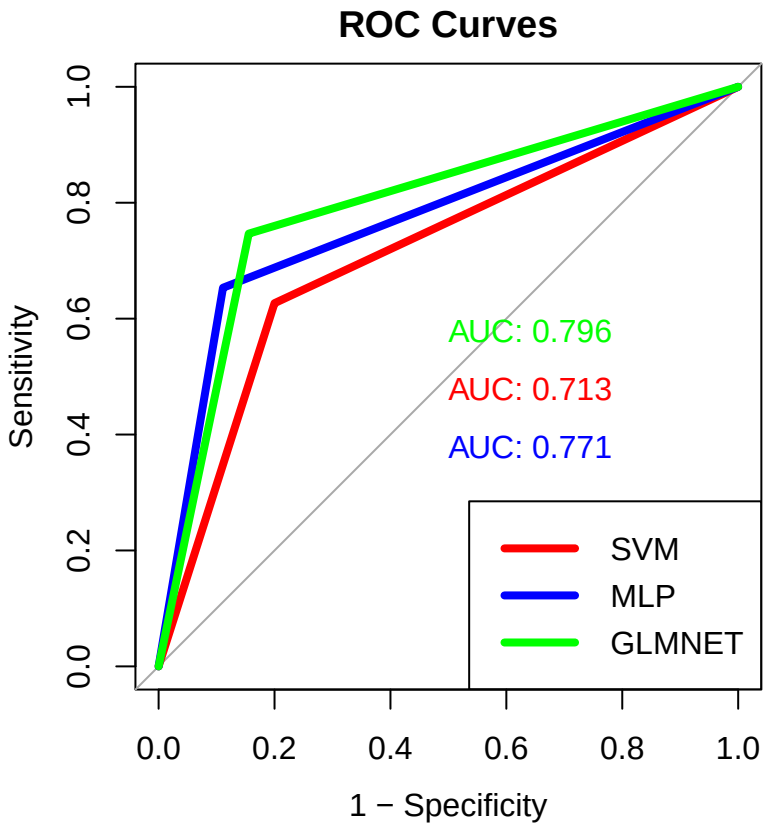


This code creates ROC curves for each model with different colours. We then print the AUC values for each curve.

ROC Curves (WoW dataset)

Non-tuned models

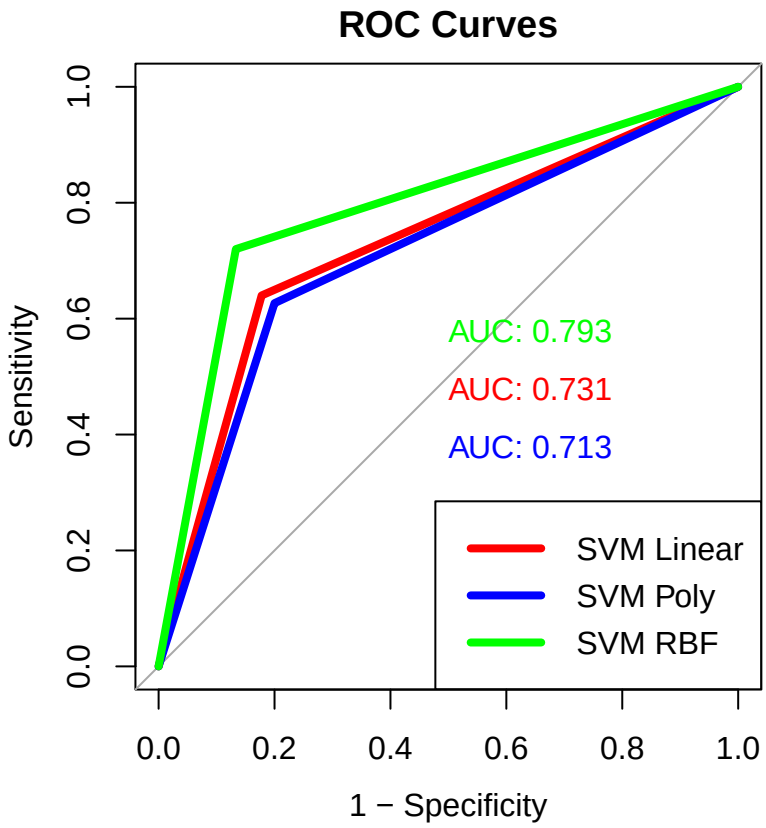
```
plot_roc_curves(svm_wow_pred_y, mlp_wow_pred_y, glmnet_wow_pred_y, c("SVM", "MLP", "GLMNET"), wow_test_l
```



It would appear that the glmnet model is very effective when used with the world of warcraft dataset as it has a much greater specificity than the other models.

Comparing tuned SVM models

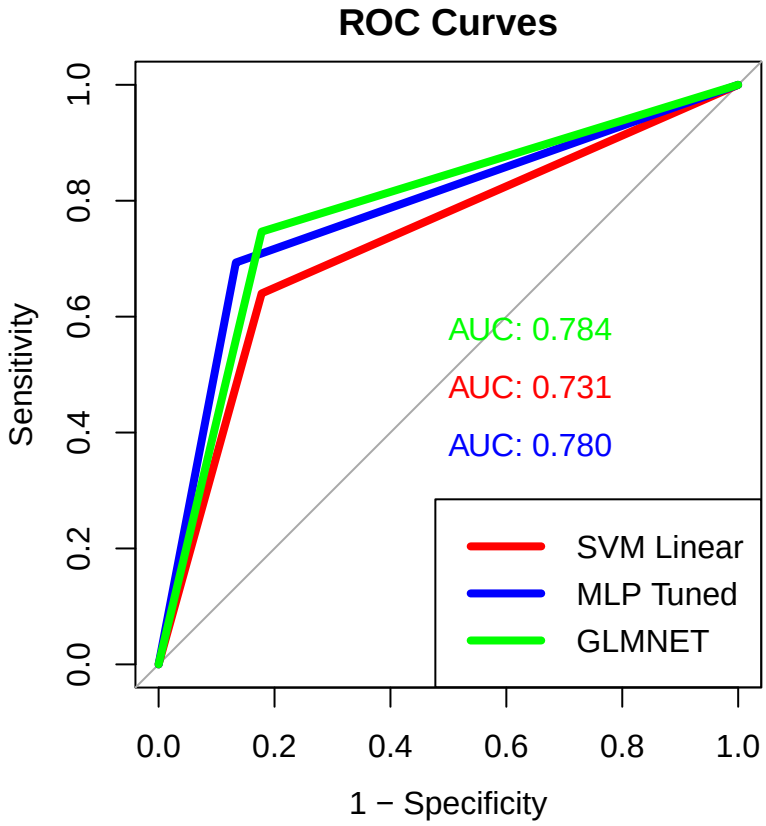
```
plot_roc_curves(svm_tuned_wow_pred_y, svm_wow_poly_pred_y, svm_wow_rbf_pred_y, c("SVM Linear", "SVM Poly
```



It would appear that the linear model has the greatest AUC. We can compare this model with other models.

Tuned models

```
plot_roc_curves(svm_tuned_wow_pred_y, mlp_wow_tuned_pred_y, glm_wow_tuned_pred_y, c("SVM Linear", "MLP
```

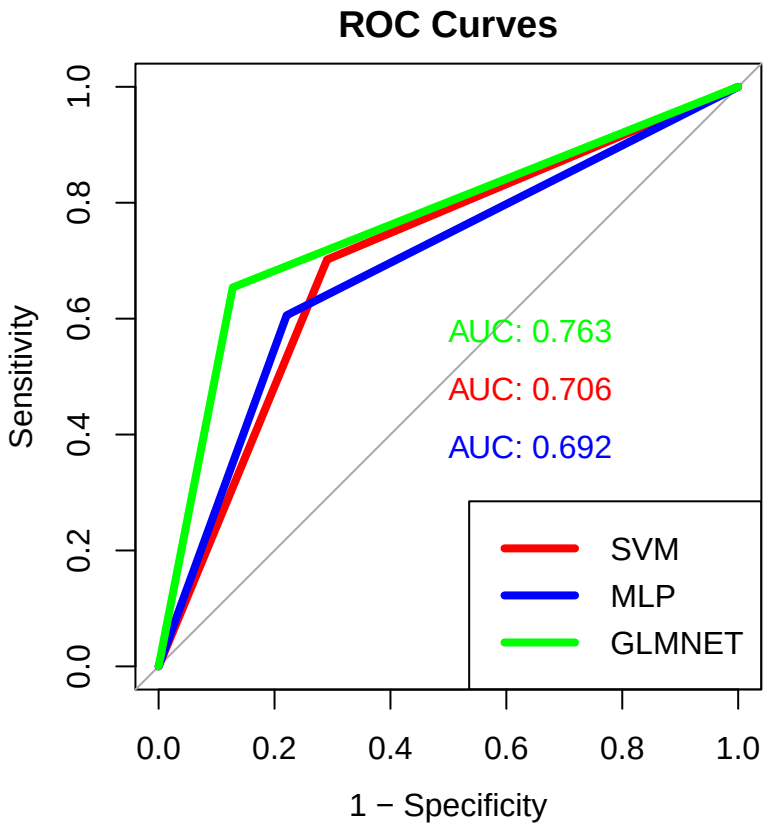


Surprisingly enough, the glmnet model has the worst performance out of all the models despite being tuned. This would suggest that the tuning had an overall negative effect.

ROC Curves (LoL dataset)

Non-tuned models

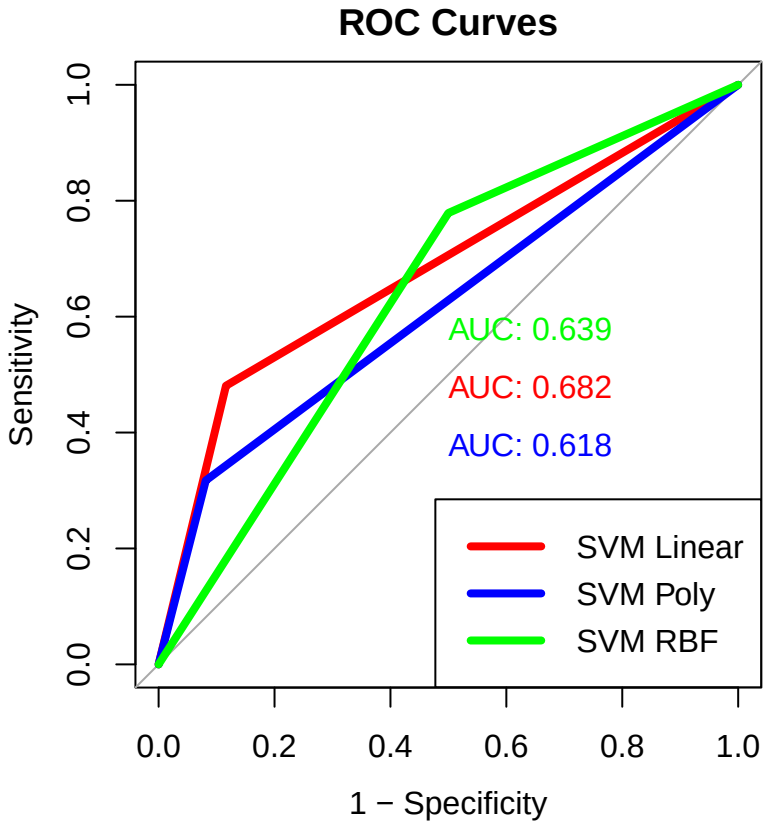
```
plot_roc_curves(svm_lol_pred_y, mlp_lol_pred_y, glmnet_lol_pred_y, c("SVM", "MLP", "GLMNET"), lol_test_l
```



It appears that glmnet consistently has good specificity when non-tuned, giving it a decent AUC score.

Comparing tuned SVM models

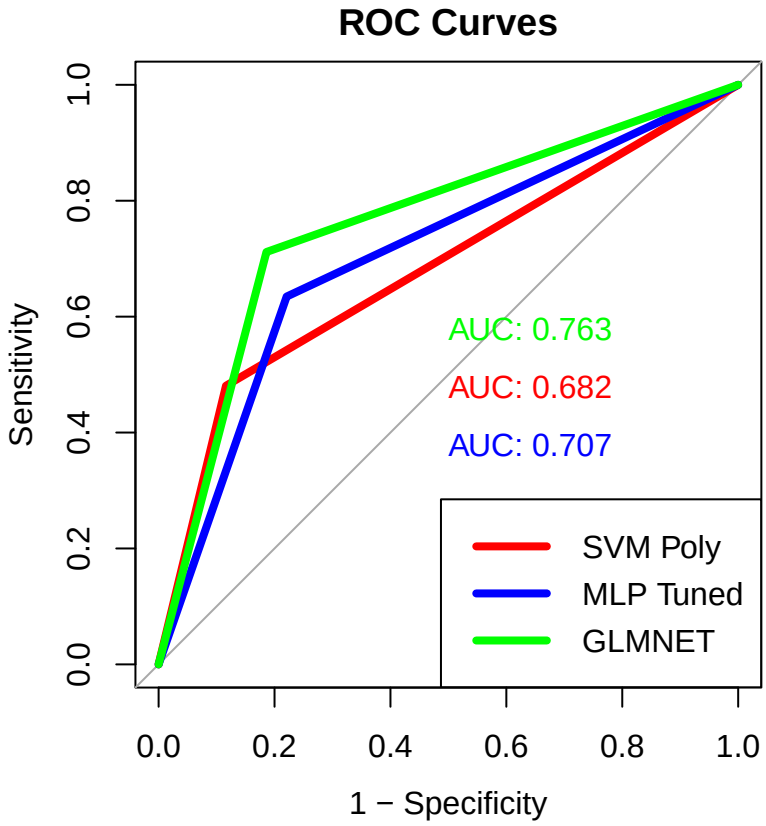
```
plot_roc_curves(svm_tuned_lol_pred_y, svm_lol_poly_pred_y, svm_lol_rbf_pred_y, c("SVM Linear", "SVM Poly
```



It would appear that the linear kernel is most suitable for bullying classification. We will compare this SVM model that uses the linear kernel with other models.

Tuned models

```
plot_roc_curves(svm_tuned_lol_pred_y, mlp_lol_tuned_pred_y, glm_lol_tuned_pred_y, c("SVM Poly", "MLP Tu
```



It would appear that MLP has the greatest AUC score when tuned.

Table of metrics

For each model, we will be calculating the evaluation metrics we will be using (time taken, precision, sensitivity, f1 score, AUC) to create a table of metrics which can be used to evaluate each model. We are acquiring the time taken to train each model to see how practical the models would be if the models were to be trained continuously with new data in terms of computation. While it doesn't take that long to train the models with the sample data, it would take much longer if there was more data.

```
create_table <- function(prediction_list, time_to_train_c, test_data) {
  Model_Name = names(prediction_list)
  Precision = c()
  Recall = c()
  F1_Score = c()
  AUC = c()
  Time_to_Train_secs = time_to_train_c

  for (prediction_y in prediction_list) {
    Precision <- append(Precision, precision(prediction_y, test_data$is_bullying))
    Recall <- append(Recall, recall(prediction_y, test_data$is_bullying))
    F1_Score <- append(F1_Score, F_meas(prediction_y, test_data$is_bullying))
    AUC <- append(AUC, auc(test_data$is_bullying, as.integer(prediction_y)))
  }
```

```
  results <- data.frame(Model_Name, Precision, Recall, F1_Score, AUC, Time_to_Train_secs)
  return(results)
}
```

Combined dataset table

```
prediction_list = list(svm_linear = svm_tuned_pred_y, svm_poly = svm_poly_pred_y, svm_rbf = svm_rbf_pred
time_to_train = c(svm_time_taken, svm_poly_tuned_time_taken, svm_rbf_tuned_time_taken, mlp_time_taken, g
results = create_table(prediction_list, time_to_train, test_balanced)
results
```

```
##   Model_Name Precision    Recall  F1_Score       AUC Time_to_Train_secs
## 1 svm_linear 0.7500000 0.8684211 0.8048780 0.7949700               5.72
## 2   svm_poly 0.7351351 0.8947368 0.8071217 0.7923051               9.25
## 3    svm_rbf 0.7383721 0.8355263 0.7839506 0.7753581              11.02
## 4        mlp 0.7382550 0.7236842 0.7308970 0.7384244             119.50
## 5     glmnet 0.6919431 0.9605263 0.8044077 0.7745670               6.70
```

From these results, it would appear that the glmnet logistic regression model has the greatest performance as it has the greatest F1 score and AUC score. It is also relatively quick to train which can reduce costs if deployed in a large-scale system where the model may be continuously trained with new data. However, the SVM model with a polynomial kernel appears to be slightly better in terms of recall than the glmnet model. However, this difference is very small to where it won't really amount to anything.

We can also see that the mlp model performed the worst as it took a long time to train on a relatively small sample of data (almost 2 minutes) and it has the worst F1 and AUC scores.

WoW dataset table

```
prediction_list = list(svm_linear = svm_tuned_wow_pred_y, svm_poly = svm_wow_poly_pred_y, svm_rbf = svm
time_to_train = c(svm_wow_time_taken, svm_wow_poly_tuned_time_taken, svm_wow_rbf_tuned_time_taken, mlp_w
wow_results = create_table(prediction_list, time_to_train, wow_test_balanced)
results
```

```
##   Model_Name Precision    Recall  F1_Score       AUC Time_to_Train_secs
## 1 svm_linear 0.7500000 0.8684211 0.8048780 0.7949700               5.72
## 2   svm_poly 0.7351351 0.8947368 0.8071217 0.7923051               9.25
## 3    svm_rbf 0.7383721 0.8355263 0.7839506 0.7753581              11.02
## 4        mlp 0.7382550 0.7236842 0.7308970 0.7384244             119.50
## 5     glmnet 0.6919431 0.9605263 0.8044077 0.7745670               6.70
```

From these results, we can see that the svm_linear model has the greatest f1 score (69%) and the greatest AUC score. Compared to the combined datasets table, glmnet performs significantly worse which could suggest that glmnet works significantly better with larger datasets.

LoL Dataset Table

```
prediction_list = list(svm_linear = svm_tuned_lol_pred_y, svm_poly = svm_lol_poly_pred_y, svm_rbf = svm
time_to_train = c(svm_lol_time_taken, svm_lol_poly_tuned_time_taken, svm_lol_rbf_tuned_time_taken, mlp_
lol_results = create_table(prediction_list, time_to_train, lol_test_balanced)
lol_results
```

```
##   Model_Name Precision    Recall  F1_Score       AUC Time_to_Train_secs
## 1 svm_linear 0.5846154 0.8837209 0.7037037 0.6822451               1.94
## 2   svm_poly 0.5266667 0.9186047 0.6694915 0.6179562               5.64
## 3    svm_rbf 0.6515152 0.5000000 0.5657895 0.6394231               6.69
## 4        mlp 0.6380952 0.7790698 0.7015707 0.7068426              94.99
## 5     glmnet 0.7000000 0.8139535 0.7526882 0.7627460               3.89
```

From these results, we can see that the tuned MLP model performs the greatest for the League of Legends dataset. However, it takes a long time to train which could prove computationally expensive when provided with more data. We can see a bias towards a higher recall versus precision when comparing the precision values to the recall values.

Output results

```
write.csv(lol_results, "Data/lol_results.csv")
write.csv(wow_results, "Data/wow_results.csv")
write.csv(results, "Data/results.csv")
```

We output the table results to a csv file to make it easy to share results with other group members.

Individual contributions

Thomas Newton

I was responsible for predictive modelling and hyperparameter tuning. I also created the problem statements and references.

Leon Harper

I was responsible for importing the data (converting it from an SQL Dump file into corresponding csv files), cleaning the data and evaluating the performance of relevant models. I helped manage the project through the use of a trello board and I helped individuals on the project who were struggling.

Michal Jedruszczak

In this project I was responsible for the preliminary EDA and the final EDA. I also created the contents section and made sure that all of the references were in MMU Harvard format.

References

Anti-Defamation League (2022) *Hate is No Game: Harassment and Positive Social Experiences in Online Games 2021* [Online][Accessed: November 19, 2022] https://www.adl.org/resources/report/hate-no-game-harassment-and-positive-social-experiences-online-games-2021

Bretschneider, U. and Peters, R. "Detecting Cyberbullying in Online Communities" (2016). *Research Papers*. Paper 61. http://aisel.aisnet.org/ecis2016_rp/61