focus effort state victory bonus one buying higher ghost worth every stand created current cities isolation immersion customize score things ii yeah orders make wanted races second troops range simon gets throughout perfectly either thanks lets top times still battles something series years sea early etc fallen eventually armies detail normal personal level hero seems castle barbarian breaking capitol near wall plan run completely lighting limbo strategic already information door review allow bad adds big heroes alien experienced black points knowing jack come except fi sound mumble crashes actions complete trying limited total regions linear enemies best horror developer tree damage pc spent resource liked able single xcom genre later possible options less want playing people endless nowhere low seem fantastic m least factor high nothing encounter overall several use large immersive expect indie gandhi love devs however filled improved straight sci meow s stack cause village awesome short say although avoid four third understand games jordan another land simple feeling fps animations unless moving aside nations equipment d care given gives population access massive side time per upgrade remember ok direction boring minutes **text** now effect doubt purchase russia ghosts hard brain felix emperor wise **mining** gorgeous classic textures masterpiece guess fun others feels form designed friend ships pants loading recommend will player work special worst starts saying story starting kill number drops protagonist help available hand leaves almost turn diversity falls lights specific lots parts played cats dead certainly batteries walls flaws sounds ever loved free example fatal ufo cover blood creepy heavy atmospheric otherwise creatures you ability area ben decided particularly replayability balance gonna ocean hit fact narrative content friends building explore behind issues can actual hide research shit squad sadly guys science terrain giant food problems empire well awesome version civilization quality outside agents mission thought tech disturbing siege great favorite tactics issue gold engine terrible opinion fight past bear go days miss forget diplomatic mods style taking cavalry main send constantly day evil god gone **by** support open highly depending challenging towards defense went slow stay **André Martinez** campaign base fair walk took certain implemented bugs depth types **Maria João Ferreira** cheap used brings personally terror performance abandoned felt simply made edit end major like visuals enemy needs sort character feel history seriously dlc group gf unknown red re keeps game love might much control within ethiopia action head follow descent capture you guys boss mean video without longer due stupid changes type useless happened shows different step fixed problem negative apart man frictional ambient map often attempt weak campaigns finding keep known suckst confusing item next call headphones sale asylum life happen gave fast maps questions players good advanced controls build resources quite finished missions shogun naval just destroy must waiting strong ideas intense lore us faction place poor adventure difference masturbate release lead choices earth similar gameplay face basic battle nation start incredible amnesia medieval looking ways wars launch whatever basically army space path x management underwater started never city events concept engaging takes rides **MADSAD** solid bonuses probably cons loss actually become usually beat developed **2015/16** kind view walking particular civ computer running upon creative characters comes storyline culture finally front aspects weapons thinking solve came monsters brilliant states features pretty rest nukes think objects capital across school encounters camera forward looks gamer true animation infantry click survival unlock happy atmosphere fantasy lot rather way tw fighting randomly sure real pick cool core machine screen back taken saw unique compared eyes multiplayer technology terrifying titles winter instance somewhat hours leave scares soon mostly technologies despite changed coming buy declare panic name forces slowly fan settlements various conclusion jump anyway fall pros robots anything soldier voice meet choice read edition away graphics dark based add patch far play definitely turns becomes lost thus crap diplomacy ended many missed reading someone details target mention making last save sorry

CONTENTS

THE DATA

For this assignment we decided to look at the Steam® platform, software and market place for computer games, and use the reviews made inside the platform by users. This feature of the platform is one of the reasons it became such a huge business success in the last few years: these are reliable and easily accessible peer reviews that can be filtered in many different ways.

We decided to use the Text Mining tools to apply to reviews of games of two different genres and check if an algorithm could classify their genres correctly. We realized we needed a substantial sample of reviews, so we chose games based on their popularity and therefore number of reviews. We would be then focusing on those reviews that got the best ratings from users.

When it came to selecting which genre we should pick, we decided to go for two which were significantly different and within which there was no overlap. We chose the user defined tags 'Psychological Horror' and 'Turn Based Strategy'(TBS), for which we couldn't find any games with both tags. This choice should make the classification task somewhat easier. The games that made up our sample were, for 'Turned based Strategy': XCOM: Enemy Unknown, Sid Meier's Civilization® V, Total War™: ROME II - Emperor Edition and Endless Legend. For 'Psychological Horror' we picked Amnesia, Outlast, SOMA and DreadOut.

We gathered the sample ourselves, filtering the reviews, as planned, by the most useful (based on user ratings) of all time, and also enforcing a rule that the review should not be shorter than a 'Tweet'. This meant that no review in our sample, even if voted as very useful by users, would be composed of less than 140 characters. Our sample was made of 300 reviews from each genre, composed of 75 reviews from four of the most popular games. In order to do this, we had to actually copy/paste each review by hand because of the way the Steam® website works: it loads only about 30 reviews per page, so the user has to call out to load more. This made it impossible for us to automatize this process with R or other tools we knew.

The fact we gathered the data "by hand" did allow us to notice some patterns in the sample right away. One difference between samples from 'Horror' and from 'Strategy' was remarkable: in the 'Horror' genre, there were many reviews marked as most useful (of all time) that were very short, sarcastic and more "funny", but they weren't very common for 'Strategy'. The most useful reviews in TBS were often longer and therefore more detailed, meaning they had many more words in general. This made us suspect that the latter reviews had better 'quality'.

There were several challenges particular to this data that we had to overcome. First, Steam® enforces censorship of swear words by replacing them with emoticons of hearts: ♥. Well, R does not recognize this character, so we had to look into the data and check what R was reading when it came across this, and found out it read "â™¥". Another problem was the usage of dashes mid-word that would become a problem once we removed punctuation. A term like "mid-word" would then become "midword", which is not the same as "mid" or "word" and not counted as such.

We also suspect that a lot of the users that wrote the reviews on Steam did so previously on a word processor, which applies changes to certain characters like quotation marks and ellipsis. These so called "smart quotes" are not handled well by R, which does not read them as punctuation but as characters (as we saw with the emoticons).

Another issue we discovered was that users did not write the title of one of the games in a similar way. The game 'XCOM' could appear with or without a space separating the "X" from "COM" (since we already removed the dash), and we wanted to make sure that it always showed up in the same manner. In order to do this, we corrected all instances with a space to without one.

All of these situations had to be corrected, as well as others we found out along the way, and we did this by creating the function that will follow.

We chose to also to divide the sample in test and train in this step, making a selection of about 30% into test and 70% to training. We had to round this percentage because we had 4 games per genre and we wanted to balance both data sets. We divided the sample randomly, entering the seed as an argument in the function, which enabled us to make this step several times easily and test the algorithm for different organizations of the data.

```
preprocessing <- function(seed = 6) {
    set.seed(seed)
    genres <- list.files("./dados-originais")
```

```
    sampling <- c(rep(1, 52), rep(0, 23))
    for (genre in genres) {
        games <- list.files(file.path("./dados-originais/", genre, fsep = ""))
        for(game in games) {
            reviews1 <- list.files(file.path("./dados-originais/", genre, "/", game, fsep = ""))
            i <- 0
            ordering <- sample(sampling, size = 75, replace = FALSE)
            for(steam_id in reviews1) {
                content <- readLines(file.path("./dados-originais/", genre, "/", game, "/", steam_id, fsep =
"", warn = FALSE)
                content <- gsub("-", " ", content)
                content <- gsub("â™¥", " ", content)
                content <- gsub("â€“", " ", content)
                content <- gsub("â€™", "'", content)
                content <- gsub("â€œ", "\"", content)
                content <- gsub("â€\u008d", "", content)
                content <- gsub("â€\u009d", "\"", content)
                content <- gsub("â˜…", "", content)
                content <- gsub("â˜†", "", content)
                content <- gsub("â€˜", "\'", content)
                content <- gsub("X COM", "XCOM", content)
                content <- gsub("x com", "xcom", content)
                i <- i + 1
                if(ordering[i] == 1) {
                    writeLines(content, file.path("./Data - TM/", genre, " train/", paste(game, steam_id), fsep
= ""))
                }
                else if(ordering[i] == 0) {
                    writeLines(content, file.path("./Data - TM/", genre, " test/", paste(game, steam_id), fsep
= ""))
                }
            }
        }
    }
}
```

After this step was done, with the sample reorganized and “clean” of these artificial contents, we wrote further lines of code to check the word frequency before converting the sample using the *tm* package. We wanted to check which words were repeated overall in the sample that were irrelevant for our analysis, and words specific of the genres that could potentially artificially give us some better results. With this we mean words like “game”, “play”, etc., the names of the games:

```
horror.train <- list()
for(i in 1:length(hor.train)) {
    new <- c()
    for(j in 1: length(hor.train[[i]][[1]])) {
        new <- c(new, unlist(strsplit(hor.train[[i]][[1]][j], " ")))
    }
    new <- new[-which(new == "")]
    horror.train <- c(horror.train, list(new))
}
baghorror <- unlist(horror.train)
sort(table(baghorror),decreasing = TRUE)[1:50]

strategy.train <- list()
for(i in 1:length(strat.train)) {
    new <- c()
    for(j in 1: length(strat.train[[i]][[1]])) {
        new <- c(new, unlist(strsplit(strat.train[[i]][[1]][j], " ")))
    }
    new <- new[-which(new == "")]
    strategy.train <- c(strategy.train, list(new))
}
bagstrat <- unlist(strategy.train)
sort(table(bagstrat),decreasing = TRUE)[1:50]
```

After running this function, we added to the stop-words the following list of what we called useless-words:

```
useless.words <- c(stopwords(kind = "en"), "one", "game", "can", "play", "will", "like", "get", "time",
"make", "just", "also", "even", "amnesia", "outlast", "soma", "dreadout", "take", "good", "look", "endless",
"total", "sid", "meier", "meiers", "legend", "rome", "xcom", "civ", "horror", "strategy", "turn", "based",
"gameplay")
```

USAGE OF THE TM PACKAGE

In order to read the reviews into R and further pre-process them, we used the *tm* package. Since our texts were divided into 4 folders (test and train for both 'Horror' and 'Strategy') we ran the lines of code below for each group.

```
hor.train <- Corpus(DirSource("Data - TM/Horror train"), readerControl=list(reader =
readPlain,language="en"))
hor.train <- tm_map(hor.train, content_transformer(tolower))
hor.train <- tm_map(hor.train, removeWords, useless.words)
hor.train <- tm_map(hor.train, removePunctuation)
hor.train <- tm_map(hor.train, removeNumbers)
hor.train <- tm_map(hor.train, stemDocument)
hor.train <- tm_map(hor.train, removeWords, useless.words)
hor.train <- tm_map(hor.train, stripWhitespace)
```

We made some modifications to the base code presented in the slides provided. First, we set the argument *readerControl* to *readPlain*, since the option in the slides would remove the first lines of the text. We then converted all letters to lower case. Having done this, we remove the stop-words defined before, followed by the removal of punctuation and numbers. After this step we stem the document. At this point we run the line to remove words again, and finish up by removing instances of multiple whitespaces.

The reason why we ran the code to remove words twice has to do with the list of words. We remove stop-words before removing punctuation, since that list contains words with punctuation (e.g. "we're") which would not be deleted if we removed the punctuation first. However, later on we stem the document, causing words like "games" to become "game", which is in our list of useless words, and so we have to run the code again to make sure we remove these words as well.

After doing this, we join the training sets together and do the same for the test sets:

```
reviews <- c(hor.train, strat.train)
testreviews <- c(hor.test, strat.test)
```

DOCUMENT TERM MATRIX

When converting our sets into document term matrices we decided to keep the default values for the option *control*, except for the weighting. The *control* option defines the weighting function that is used for the terms, the character length of terms and the bounds, which accounts for in how many documents a word has to appear in order to be considered for the document term matrix. Regarding word length, the default is 3 characters, which fits our sample, since the reviews have a lot of short frequent words like "war", "map" and "run" and we obviously want to keep those. The stemming process did reduce terms by quite a lot as well, so at least 3 characters seems to be fitting.

We used the default bounds, 1, since later on we used a sparsity filter, which will instead enforce a minimal percentage of documents in which the terms have to appear in order to be considered, rendering this option useless at this time.

Regarding the weighing function, we did try two options, default (*tf*) and *tfidf* which stands for term frequency-inverse document frequency, that basically gives a numerical statistic to a term that reveals how important that term is inside the document. The value will increase with the proportion that the term is repeated in the document, but is offset by the frequency in which it appears in the whole sample.

FEATURE REDUCTION

To reduce the number of features in our data sets we used sparsity correction: we removed terms that did not appear in many documents. This will not only eliminate words that barely appear in all the documents, but also those that show up a lot of times in very few different reviews.

We decided to test 2 different values for the sparsity term, which we will call S. S is the maximum sparsity allowed: this means that, for a sparsity level of 95% (for example), we will remove words that don't appear in at least 5% of the documents. The higher the value of S, the more terms will be kept since we are allowing words with a lesser document frequency to stay in the data set.

The two values we used for *S* were 90% and 95%. We combined these with each of the two weighting options, resulting in 4 data sets. A value of 90% for *S* reduced the number of terms from 5780 to 156 and a value of 95% reduced the number of terms to 454. The type of weight did not affect the number of attributes nor the attributes themselves.

The code to generate the data set with *tf* weights and a sparsity level of 95% was:

```
#Training set:
dtm95 <- as.data.frame(inspect(removeSparseTerms(dtm, 0.95)))
dtm95 <- cbind(dtm95, genre_class = c(rep("horror", 208), rep("strategy", 208)))

#Test set:
dtm.te <- DocumentTermMatrix(testreviews)
terms95 <- names(dtm95)[-length(names(dtm95))]
dtm.te95 <- dtm.te[,terms95]
dtm.te95 <- as.data.frame(inspect(dtm.te95))
dtm.te95 <- cbind(dtm.te95, genre_class = c(rep("horror", 92), rep("strategy", 92)))
```

FEATURE SELECTION

For feature selection we used information gain. We calculated this measure using the *information.gain* function in the *FSelector* package.

We chose to test three different levels of minimum information gain: keeping all the variables with an information gain above 0, 0.03 and 0.05. We chose the level of 0.03 because most variables with a positive information gain were above this value, so only a few would be eliminated. The 0.05 would eliminate a more considerable number of variables. Even an information gain above 0 was enough to severely reduce the number of attributes. The reduction of the number of terms went as follows:

	IG > 0		IG > 0.03		IG > 0.05	
	TF	TFIDF	TF	TFIDF	TF	TFIDF
Sparsity = 100%: 5780 terms	95	111	47	54	19	21
Sparsity = 95%: 454 terms	57	65	41	47	19	21
Sparsity = 90%: 156 terms	22	25	18	20	14	16

We did a few tests for these different levels of minimum information gain, which we will talk about in the section about the classification models.

After creating all these different data sets, we added a column to the end of each with the true genre of the game the review is about, which is our target variable. We then created a different test data set for each of our training sets in order to include the same terms and weight measures of the respective training sets.

The code for one training set (*tf* weights, information gain above 0, sparsity level of 95%) and its respective test set is as follows:

```
##Training set:
dtm <- DocumentTermMatrix(reviews)
#Sparse terms
dtm.ig95 <- as.data.frame(inspect(removeSparseTerms(dtm, 0.95)))
#Target variable column
dtm.ig95 <- cbind(dtm.ig95, genre_class = c(rep("horror", 208), rep("strategy", 208)))
#Information gain
library(FSelector)
dtm.ig95 <- dtm.ig95[, c(which(information.gain(genre_class ~., dtm.ig95)$attr_importance != 0),
ncol(dtm.ig95))]

##Test set:
dtm.te <- DocumentTermMatrix(testreviews)
terms.ig95 <- names(dtm.ig95)[-length(names(dtm.ig95))]
dtm.te.ig95 <- dtm.te[,terms.ig95]
dtm.te.ig95 <- as.data.frame(inspect(dtm.te.ig95))
dtm.te.ig95 <- cbind(dtm.te.ig95, genre_class = c(rep("horror", 92), rep("strategy", 92)))
```

CLASSIFICATION MODELS

As for the models, we tried all the main models we learned in classes, trying different settings and with the different data frames that resulted from the several methods we used on the pre-processing phase. Therefore, we will show now the results and analysis of the classification running K-Nearest Neighbours, Decision Trees, Naïve Bayes, Neural Networks and Support Vector Machines.

To train classifiers on our data sets we used the *caret* package. This package accesses the algorithms in other packages (such as *rpart*), and allows us to test several different types of models by performing minimal changes to the code. By default, it will also do the bootstrapping sampling method as a cross validation method (25 repetitions), providing a relatively reliable accuracy result from the training set.

However, the Naïve Bayes method gave us too many errors, resulting in a very poor classifier. As an alternative we used the *RWeka* package for this classifier. Since we were not working with the *caret* package, we could not use cross-validation for this method, as bootstrapping is a complicated procedure to apply. We could do easier methods of cross-validation (such as hold-out cross validation), but the results would not be as reliable as bootstrapping and thus not comparable to the remaining models. We decided to simply train and predict on the entire training set to have a general idea of the accuracy of the model, while being aware that those results are not reliable: overfitting is likely to happen.

Before creating each model, we set the seed to 5465, so that the bootstrap samples would be the same during the training process. Below is an example of the generation of a decision tree model with *tf* weights and $S = 90\%$. The first argument of the *train* function are the attributes, the second argument is the target attribute, and the third is the method. Additional arguments may be passed to fine-tune the model.

```
library(caret)
set.seed(5465)
mo.dt90 <- train(dtm90[,-ncol(dtm90)], dtm90[, ncol(dtm90)], method = "rpart")
mo.dt90
#CART
#
#416 samples
#156 predictors
#  2 classes: 'horror', 'strategy'
#
#No pre-processing
#Resampling: Bootstrapped (25 reps)
#Summary of sample sizes: 416, 416, 416, 416, 416, 416, ...
#Resampling results across tuning parameters:
#
#  cp          Accuracy   Kappa      Accuracy SD  Kappa SD
#  0.03846154  0.7778725  0.5554588  0.04385527   0.08590830
#  0.08653846  0.7174527  0.4407173  0.04195129   0.08077645
#  0.38461538  0.5771485  0.1869498  0.10604375   0.17326699
#
#Accuracy was used to select the optimal model using the largest value.
#The final value used for the model was cp = 0.03846154.
```

For the Naïve Bayes model we had to run the following code:

```
library(RWeka)
NB <- make_Weka_classifier("weka/classifiers/bayes/NaiveBayes")
set.seed(5465)
mo.nb90 <- NB(genre_class ~., dtm90)
```

In order to make the process of generation and evaluation of the models for each data set faster, we created a function in R that would allow us to run different algorithms for different data sets with minimal changes to the arguments of the said function. It will load the necessary libraries, create the requested model using the defined training and test sets, and will print performance evaluators for both sets.

It has 5 arguments, the first two being *train* and *test*, which are the training and test sets, respectively. The third argument is the *method*, which is the classifier we wish to generate. We can use any method accepted by the *train* function in the *caret* package (full list can be found here). For the classifiers we decided to test, the methods are: "*rpart*" for decision trees, *"knn"* for nearest neighbours, *"nb"* for Naïve Bayes (note that this one will not use the *caret* package), *"nnet"* for neural networks

and *"svmRadial"* for SVM's. The function has two other optional arguments: the *seed* and *return_model*, which allows us to save the model to an object when *TRUE*. These have default values and don't need to be specified unless necessary.

```
run.model <- function(train, test, method, seed = "5465", return_model = FALSE) {
    if(method == "nb") {
        library(RWeka)
        set.seed(seed)
        NB <- make_Weka_classifier("weka/classifiers/bayes/NaiveBayes")
        model <- NB(genre_class ~., train)
        cat("\n", "Training set confusion matrix:", sep = "")
        print(table(train[, ncol(train)], predict(model, train[,-ncol(train)])))
        cat("\n", "Test set confusion matrix:", sep = "")
        print(table(test[, ncol(test)], predict(model, test[,-ncol(test)])))
    } else {
        library(caret)
        set.seed(seed)
        model <- train(train[,-ncol(train)], train[, ncol(train)], method = method)
        print(model)
        if(method == "knn") set.seed(seed)
        cat("\n", "Test set confusion matrix:", sep = "")
        print(table(test[, ncol(test)], predict(model, test[,-ncol(test)])))
    }
    if(return_model == TRUE) return(model)
}
```

For example, to create the same models we did above using *rpart* and Naïve Bayes we can do the following:

```
mo.dt90 <- run.model(dtm90, dtm.te90, "rpart", return_model = T)
mo.nb90 <- run.model(dtm90, dtm.te90, "nb", return_model = T)
```

This will not only do the same as the lines of code from before, but it will also make the predictions for the test set and generate the respective confusion matrix. Using the above decision tree as an example:

```
mo.dt90 <- run.model(dtm90, dtm.te90, "rpart", return_model = T)
# CART

# 416 samples
# 156 predictors
  # 2 classes: 'horror', 'strategy'

# No pre-processing
# Resampling: Bootstrapped (25 reps)
# Summary of sample sizes: 416, 416, 416, 416, 416, 416, ...
# Resampling results across tuning parameters:

  # cp          Accuracy   Kappa      Accuracy SD  Kappa SD
  # 0.03846154  0.7778725  0.5554588  0.04385527   0.08590830
  # 0.08653846  0.7174527  0.4407173  0.04195129   0.08077645
  # 0.38461538  0.5771485  0.1869498  0.10604375   0.17326699

# Accuracy was used to select the optimal model using  the largest value.
# The final value used for the model was cp = 0.03846154.

# Test set confusion matrix:
           # horror strategy
  # horror       56       36
  # strategy      2       90
```

As we can observe, this command prints the same information as before, plus the confusion matrix of the test set.

PERFORMANCE OF THE MODELS

We will now present the performance measures of our 5 types of classifiers to our different data sets, excluding those that used information gain as feature selection. The accuracy obtained in the training set is via cross validation, namely bootstrapping. The only exception is the Naïve Bayes model, for the reasons already mentioned. We used the values returned by the confusion matrices to calculate the remaining performance measures based on the test set.

dtm90	Accuracy											Confusion Matrix			
method	Train	Test	MacroF1	PrecH	RecH	F1-H	PrecS	RecS	F1-S	MacroP	MacroR	TP	FN	FP	TN
rpart	0.778	0.793	0.816	0.966	0.609	0.747	0.714	0.978	0.826	0.840	0.793	56	36	2	90
knn	0.712	0.826	0.833	0.773	0.924	0.842	0.905	0.728	0.807	0.839	0.826	85	7	25	67
nb	0.808	0.799	0.818	0.724	0.967	0.828	0.951	0.630	0.758	0.837	0.799	89	3	34	58
nnet	0.863	0.859	0.859	0.844	0.880	0.862	0.875	0.837	0.856	0.859	0.859	81	11	15	77
svmRadial	0.751	0.810	0.820	0.913	0.685	0.783	0.748	0.935	0.831	0.830	0.810	63	29	6	86

dtm95	Accuracy											Confusion Matrix			
method	Train	Test	MacroF1	PrecH	RecH	F1-H	PrecS	RecS	F1-S	MacroP	MacroR	TP	FN	FP	TN
rpart	0.768	0.793	0.816	0.966	0.609	0.747	0.714	0.978	0.826	0.840	0.793	56	36	2	90
knn	0.748	0.810	0.827	0.736	0.967	0.836	0.952	0.652	0.774	0.844	0.810	89	3	32	60
nb	0.827	0.755	0.769	0.694	0.913	0.789	0.873	0.598	0.710	0.784	0.755	84	8	37	55
nnet	0.893	0.886	0.886	0.874	0.902	0.888	0.899	0.870	0.884	0.886	0.886	83	9	12	80
svmRadial	0.773	0.804	0.817	0.924	0.663	0.772	0.737	0.946	0.829	0.831	0.804	61	31	5	87

dtm.idf90	Accuracy											Confusion Matrix			
method	Train	Test	MacroF1	PrecH	RecH	F1-H	PrecS	RecS	F1-S	MacroP	MacroR	TP	FN	FP	TN
rpart	0.771	0.739	0.777	0.978	0.489	0.652	0.659	0.989	0.791	0.819	0.739	45	47	1	91
knn	0.788	0.826	0.842	0.969	0.674	0.795	0.750	0.978	0.849	0.859	0.826	62	30	2	90
nb	0.916	0.935	0.935	0.955	0.913	0.933	0.917	0.957	0.936	0.936	0.935	84	8	4	88
nnet	0.895	0.908	0.908	0.912	0.902	0.907	0.903	0.913	0.908	0.908	0.908	83	9	8	84
svmRadial	0.837	0.908	0.908	0.912	0.902	0.907	0.903	0.913	0.908	0.908	0.908	83	9	8	84

dtm.idf95	Accuracy											Confusion Matrix			
method	Train	Test	MacroF1	PrecH	RecH	F1-H	PrecS	RecS	F1-S	MacroP	MacroR	TP	FN	FP	TN
rpart	0.771	0.739	0.777	0.978	0.489	0.652	0.659	0.989	0.791	0.819	0.739	45	47	1	91
knn	0.830	0.864	0.876	0.791	0.989	0.879	0.986	0.739	0.845	0.888	0.864	91	1	24	68
nb	0.950	0.935	0.935	0.944	0.924	0.934	0.926	0.946	0.935	0.935	0.935	85	7	5	87
nnet	0.928	0.951	0.951	0.956	0.946	0.951	0.946	0.957	0.951	0.951	0.951	87	5	4	88
svmRadial	0.858	0.935	0.935	0.944	0.924	0.934	0.926	0.946	0.935	0.935	0.935	85	7	5	87

We can see from these tables that the decision trees were the models that performed the worse for all of our data sets, never achieving an accuracy of 80%. They were very good classifying 'Strategy' reviews but poor for 'Horror'. The k-nearest neighbours and Naïve Bayes (for *tf* weights) algorithms were the opposite, classifying 'Horror' well but not 'Strategy'. It is worth noting that the Naïve Bayes and SVM algorithms performed much better with *tfidf* weights. The neural networks models were the best overall, achieving even better results with *tfidf* weights as well. The best results here were with *tfidf* weights and 95% sparsity, particularly with the Naïve Bayes, SVM and Neural Networks algorithms.

At this point we decided to compare the misclassified reviews of each of the best performing models, to see which reviews were not being classified properly. We ran the following code to discover which texts had been incorrectly classified by the respective model (in this example, the neural network for 95% sparsity and *tfidf* weights, no information gain):

```
rownames(dtm.te.idf95[which(dtm.te.idf95$genre_class != predict(mo.nnet.idf95,
                                                                dtm.te.idf95[,-ncol(dtm.te.idf95)])) , ])
```

We verified that the algorithms had some incorrect classifications in common. This lead us to conclude that combining multiple models would not improve the performance significantly and any increases in accuracy could be more due to overfitting than actual prediction ability.

INFORMATION GAIN

As for the data sets that were created with information gain conditions, we tested three scenarios: information gain above 0, above 0.03 and above 0.05. The decision trees always got the same predictions and accuracies, which were still the worse. The k-NN algorithm improved with an increased value of minimum information gain, particularly with *tfidf* weights. The Naïve Bayes models with *tfidf* weights had generally the best accuracy with information gain above 0 and above 0.03. The neural networks had similar results for information gain above 0 and 0.03, and lower results for 0.05. The SVM had better results for an information gain of above 0.03 and 0. With this in mind, and given what we observed about the number of terms with each level of information gain, we can conclude that the best number of attributes for this data is between 40 and 70.

The table below summarizes the main results (accuracy):

		TF			TFIDF		
	method	IG > 0	IG > 0.03	IG > 0.05	IG > 0	IG > 0.03	IG > 0.05
S = 90%	rpart	79.35%	79.35%	79.35%	73.91%	73.91%	73.91%
	knn	86.96%	86.96%	86.96%	86.96%	89.13%	**90.76%**
	nb	85.33%	84.78%	84.24%	91.85%	92.93%	91.30%
	nnet	**94.02%**	92.39%	90.22%	**94.02%**	**94.02%**	92.39%
	svmRadial	84.24%	**88.59%**	86.41%	89.67%	**91.30%**	**91.30%**
S = 95%	rpart	79.35%	79.35%	79.35%	73.91%	73.91%	73.91%
	knn	85.33%	**88.59%**	**88.04%**	87.50%	89.67%	**90.22%**
	nb	83.70%	86.41%	86.41%	91.85%	**93.48%**	91.85%
	nnet	**94.57%**	**94.02%**	90.76%	**96.20%**	**96.20%**	92.39%
	svmRadial	87.50%	**89.67%**	85.87%	**92.39%**	91.85%	91.85%

Since the information gain above 0.03 provided the best results in most cases, we decided to use this level to make the next comparisons.

NO INFORMATION GAIN VERSUS INFORMATION GAIN > 0.03

	S = 90%				S = 95%			
	TF		TFIDF		TF		TFIDF	
	No IG	IG > 0.03	No IG	IG > 0.03	No IG	IG > 0.03	No IG	IG > 0.03
rpart	79.35%	79.35%	73.91%	73.91%	79.35%	79.35%	73.91%	73.91%
knn	82.61%	**86.96%**	82.61%	**89.13%**	80.98%	**88.59%**	86.41%	**89.67%**
nb	79.89%	**84.78%**	**93.48%**	92.93%	75.54%	**86.41%**	93.48%	93.48%
nnet	85.87%	**92.39%**	90.76%	**94.02%**	88.59%	**94.02%**	95.11%	**96.20%**
svmRadial	80.98%	**88.59%**	90.76%	**91.30%**	80.43%	**89.67%**	**93.48%**	91.85%

From this table we can see that having a condition of minimum information gain of 0.03 leads to better results, especially when using *tf* weights. For *tfidf* weights, the difference is only noticeable for the k-NN algorithm.

TF VERSUS TFIDF

	S = 90%				S = 95%			
	No IG		IG > 0.03		No IG		IG > 0.03	
	TF	TFIDF	TF	TFIDF	TF	TFIDF	TF	TFIDF
rpart	**79.35%**	73.91%	**79.35%**	73.91%	**79.35%**	73.91%	**79.35%**	73.91%
knn	82.61%	82.61%	86.96%	**89.13%**	80.98%	**86.41%**	88.59%	89.67%
nb	79.89%	**93.48%**	84.78%	**92.93%**	75.54%	**93.48%**	86.41%	93.48%
nnet	85.87%	**90.76%**	92.39%	**94.02%**	88.59%	**95.11%**	94.02%	96.20%
svmRadial	80.98%	**90.76%**	88.59%	**91.30%**	80.43%	**93.48%**	89.67%	91.85%

As we can see, the *tfidf* weighting option leads to a significantly better performance in all levels of information gain and sparsity, and for almost all algorithms, with the exception of decision trees.

SPARSITY 90% VERSUS 95%

	TF				TFIDF			
	No IG		IG > 0.03		No IG		IG > 0.03	
	S=90%	S=95%	S=90%	S=95%	S=90%	S=95%	S=90%	S=95%
rpart	79.35%	79.35%	79.35%	79.35%	73.91%	73.91%	73.91%	73.91%
knn	**82.61%**	80.98%	86.96%	**88.59%**	82.61%	**86.41%**	89.13%	89.67%
nb	**79.89%**	75.54%	84.78%	**86.41%**	93.48%	93.48%	92.93%	**93.48%**
nnet	85.87%	**88.59%**	92.39%	**94.02%**	90.76%	**95.11%**	94.02%	**96.20%**
svmRadial	80.98%	80.43%	88.59%	**89.67%**	90.76%	**93.48%**	91.30%	91.85%

From the above table we can conclude that the sparsity correction of 95% is usually better, though in a lot of cases they both perform similarly.

WRAP-UP

We conclude that, for our data and goals, the best algorithm is the neural network. It provided outstanding results for nearly all combinations of feature reduction and feature selection methods and performed better than the remaining algorithms in most cases. The best neural network models were the ones where we used *tfidf* weighting, 95% sparsity and an information gain above 0.03.

The Naïve Bayes and SVM algorithms worked better with *tfidf* weighting, while the k-NN algorithm achieved better results with an information gain above 0.03. The decision trees had higher performances with *tf* weighting, but their accuracy was relatively low compared to the remaining models.

KNN

While using KNN model we realized that, contrary to other models, every time we ran the confusion matrix, we would get slightly different results. This happened, however, because of the way KNN deals with ties. There is a random factor when deciding which are the 'k' neighbours that are chosen is randomized when there is a tie, resulting in some variation in the results. To overcome this problem, we set a seed before running the confusion matrixes, allowing us to get constant results.

```
set.seed(5465)
table(dtm.te90$genre_class, predict(mo.knn90, dtm.te90[,-ncol(dtm.te90)]))
```

ADDITIONAL EXPERIMENTATION

For each of the variations in our pre-processing methods, we tried different seeds for all the steps, in order to check the stability of our results. As we talked about before, we did not just enter a seed when running the models, but also when we run the function that allowed us to sort the sample between test and training.

There surely were some small variations, but nothing that was really relevant, or even statistically significant. We concluded that the seed was just not relevant.

CONCLUSIONS

We are very satisfied we achieved such good results, throughout all our different tries, and we learned a lot about how to deal with text data and the challenges that it can generate. It was especially interesting to see exactly what each pre-processing procedure actually did and how that would be reflected in the results, with each try improving in most cases but also hindering other algorithms. This was useful not only to understand exactly how the algorithm works, but how it deals with different kind of datasets.

The fact we had to manually to copy the reviews onto text files allowed us to see all these little details that we weren't expecting, for example how R actually reads all the automatic formats that Word or other text processors do to text. Another thing that was very interesting as well was to see that when the confusion matrix from the algorithm K-Nearest Neighbours got rolled out, it had slightly different results every time. That made us understand how some details like the tie breaking to determine which neighbours are actually the closer ones makes a difference in the end results.

The reason we had such good results as well was because we chose such different genres of games, where there was no overlay on user-generated tags. An interesting next step would be to test these results with game genres that were more alike, or maybe where the reviews had a more analogous structure. At the start of this assignment, we were not expecting that the reviews would be this good at classifying the games and therefore were concerned with choosing such dissimilar genres.

201001429 André Martinez
200502670 Maria João Ferreira

REFERENCES

R Core Team (2016). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. URL https://www.R-project.org/.

Max Kuhn. Contributions from Jed Wing, Steve Weston, Andre Williams, Chris Keefer, Allan Engelhardt, Tony Cooper, Zachary Mayer, Brenton Kenkel, the R Core Team, Michael Benesty, Reynald Lescarbeau, Andrew Ziem, Luca Scrucca, Yuan Tang and Can Candan. (2016). caret: Classification and Regression Training. R package version 6.0-64. https://CRAN.R-project.org/package=caret

Ingo Feinerer and Kurt Hornik (2015). tm: Text Mining Package. R package version 0.6-2. https://CRAN.R-project.org/package=tm

Hornik K, Buchta C and Zeileis A (2009). “Open-Source Machine Learning: R Meets Weka.” _Computational Statistics_, *24*(2), pp. 225-232. <URL: http://doi.org/10.1007/s00180-008-0119-7>.

Piotr Romanski and Lars Kotthoff (2014). FSelector: Selecting attributes. R package version 0.20. https://CRAN.R-project.org/package=FSelector

Brazdil, Pavel (2011). Classification of Documents using Text Mining Package "tm" [PowerPoint slides]. Retrieved from https://moodle.up.pt/

Steam® reviews:

- Amnesia: The Dark Descent: http://steamcommunity.com/app/57300/reviews/?p=1&browsefilter=toprated
- DreadOut: http://steamcommunity.com/app/269790/reviews/?p=1&browsefilter=toprated
- Outlast: http://steamcommunity.com/app/238320/reviews/?p=1&browsefilter=toprated
- SOMA: http://steamcommunity.com/app/282140/reviews/?p=1&browsefilter=toprated
- Total War: ROME II - Emperor Edition: http://steamcommunity.com/app/214950/reviews/?p=1&browsefilter=toprated
- Endless Legend: http://steamcommunity.com/app/289130/reviews/?p=1&browsefilter=toprated
- XCOM: Enemy Unknown: http://steamcommunity.com/app/200510/reviews/?p=1&browsefilter=toprated
- Sid Meier's Civilization® V: http://steamcommunity.com/app/8930/reviews/?p=1&browsefilter=toprated

PRE-PROCESSING

```
preprocessing <- function(seed = 6) {
    set.seed(seed)
    genres <- list.files("./dados-originais")
    sampling <- c(rep(1, 52), rep(0, 23))
    for (genre in genres) {
        games <- list.files(file.path("./dados-originais/", genre, fsep = ""))
        for(game in games) {
            reviews1 <- list.files(file.path("./dados-originais/", genre, "/", game, fsep = ""))
            i <- 0
            ordering <- sample(sampling, size = 75, replace = FALSE)
            for(steam_id in reviews1) {
                content <- readLines(file.path("./dados-originais/", genre, "/", game, "/", steam_id, fsep =
"", warn = FALSE)
                content <- gsub("-", " ", content)
                content <- gsub("â™¥", " ", content)
                content <- gsub("â€“", " ", content)
                content <- gsub("â€™", "'", content)
                content <- gsub("â€œ", "\"", content)
                content <- gsub("â€\u008d", "", content)
                content <- gsub("â€\u009d", "\"", content)
                content <- gsub("â˜…", "", content)
                content <- gsub("â˜†", "", content)
                content <- gsub("â€˜", "\'", content)
                content <- gsub("X COM", "XCOM", content)
                content <- gsub("x com", "xcom", content)

                i <- i + 1
                if(ordering[i] == 1) {
                    writeLines(content, file.path("./Data - TM/", genre, " train/", paste(game, steam_id), fsep
= ""))
                }
                else if(ordering[i] == 0) {
                    writeLines(content, file.path("./Data - TM/", genre, " test/", paste(game, steam_id), fsep
= ""))
                }
            }
        }
    }
}

# Determining useless words:
library(tm)
hor.train <- Corpus(DirSource("Data - TM/Horror train"), readerControl=list(reader =
readPlain,language="en"))
hor.train <- tm_map(hor.train, content_transformer(tolower))
hor.train <- tm_map(hor.train, removeWords, stopwords(kind = "en"))
hor.train <- tm_map(hor.train, removePunctuation)
hor.train <- tm_map(hor.train, removeNumbers)
hor.train <- tm_map(hor.train, stemDocument)
hor.train <- tm_map(hor.train, stripWhitespace)

strat.train <- Corpus(DirSource("Data - TM/Strategy train"), readerControl=list(reader =
readPlain,language="en"))
strat.train <- tm_map(strat.train, content_transformer(tolower))
strat.train <- tm_map(strat.train, removeWords, stopwords(kind = "en"))
strat.train <- tm_map(strat.train, removePunctuation)
strat.train <- tm_map(strat.train, removeNumbers)
strat.train <- tm_map(strat.train, stemDocument)
strat.train <- tm_map(strat.train, stripWhitespace)

horror.train <- list()
for(i in 1:length(hor.train)) {
    new <- c()
     for(j in 1: length(hor.train[[i]][[1]])) {
         new <- c(new, unlist(strsplit(hor.train[[i]][[1]][j], " ")))
       }
    new <- new[-which(new == "")]
```

```
    horror.train <- c(horror.train, list(new))
}
baghorror <- unlist(horror.train)
sort(table(baghorror),decreasing = TRUE)[1:50]

strategy.train <- list()
for(i in 1:length(strat.train)) {
    new <- c()
     for(j in 1: length(strat.train[[i]][[1]])) {
         new <- c(new, unlist(strsplit(strat.train[[i]][[1]][j], " ")))
       }
    new <- new[-which(new == "")]
    strategy.train <- c(strategy.train, list(new))
}
bagstrat <- unlist(strategy.train)
sort(table(bagstrat),decreasing = TRUE)[1:50]


useless.words <- c(stopwords(kind = "en"), "one", "game", "can", "play", "will", "like", "get", "time",
"make", "just", "also", "even", "amnesia", "outlast", "soma", "dreadout", "take", "good", "look", "endless",
"total", "sid", "meier", "meiers", "legend", "rome", "xcom", "civ", "horror", "strategy", "turn", "based",
"gameplay")


# Preprocessing using the tm package:
##Horror
hor.test <- Corpus(DirSource("Data - TM/Horror test"), readerControl=list(reader = readPlain,language="en"))
hor.test <- tm_map(hor.test, content_transformer(tolower))
hor.test <- tm_map(hor.test, removeWords, useless.words)
hor.test <- tm_map(hor.test, removePunctuation)
hor.test <- tm_map(hor.test, removeNumbers)
hor.test <- tm_map(hor.test, stemDocument)
hor.test <- tm_map(hor.test, removeWords, useless.words)
hor.test <- tm_map(hor.test, stripWhitespace)

hor.train <- Corpus(DirSource("Data - TM/Horror train"), readerControl=list(reader =
readPlain,language="en"))
hor.train <- tm_map(hor.train, content_transformer(tolower))
hor.train <- tm_map(hor.train, removeWords, useless.words)
hor.train <- tm_map(hor.train, removePunctuation)
hor.train <- tm_map(hor.train, removeNumbers)
hor.train <- tm_map(hor.train, stemDocument)
hor.train <- tm_map(hor.train, removeWords, useless.words)
hor.train <- tm_map(hor.train, stripWhitespace)


###Strategy:
strat.test <- Corpus(DirSource("Data - TM/Strategy test"), readerControl=list(reader =
readPlain,language="en"))
strat.test <- tm_map(strat.test, content_transformer(tolower))
strat.test <- tm_map(strat.test, removeWords, useless.words)
strat.test <- tm_map(strat.test, removePunctuation)
strat.test <- tm_map(strat.test, removeNumbers)
strat.test <- tm_map(strat.test, stemDocument)
strat.test <- tm_map(strat.test, removeWords, useless.words)
strat.test <- tm_map(strat.test, stripWhitespace)

strat.train <- Corpus(DirSource("Data - TM/Strategy train"), readerControl=list(reader =
readPlain,language="en"))
strat.train <- tm_map(strat.train, content_transformer(tolower))
strat.train <- tm_map(strat.train, removeWords, useless.words)
strat.train <- tm_map(strat.train, removePunctuation)
strat.train <- tm_map(strat.train, removeNumbers)
strat.train <- tm_map(strat.train, stemDocument)
strat.train <- tm_map(strat.train, removeWords, useless.words)
strat.train <- tm_map(strat.train, stripWhitespace)
```

FEATURE REDUCTION

```
#Break the sample into 2 groups - train and test
reviews <- c(hor.train, strat.train)
```

```
testreviews <- c(hor.test, strat.test)

dtm <- DocumentTermMatrix(reviews)
dtm.idf <- DocumentTermMatrix(reviews, control=list(weighting = weightTfIdf))

#Sparse terms and class
dtm95 <- as.data.frame(inspect(removeSparseTerms(dtm, 0.95)))
dtm90 <- as.data.frame(inspect(removeSparseTerms(dtm, 0.90)))

dtm.idf95 <- as.data.frame(inspect(removeSparseTerms(dtm.idf, 0.95)))
dtm.idf90 <- as.data.frame(inspect(removeSparseTerms(dtm.idf, 0.90)))

dtm95 <- cbind(dtm95, genre_class = c(rep("horror", 208), rep("strategy", 208)))
dtm90 <- cbind(dtm90, genre_class = c(rep("horror", 208), rep("strategy", 208)))

dtm.idf95 <- cbind(dtm.idf95, genre_class = c(rep("horror", 208), rep("strategy", 208)))
dtm.idf90 <- cbind(dtm.idf90, genre_class = c(rep("horror", 208), rep("strategy", 208)))


#To make the test set into a data frame
dtm.te <- DocumentTermMatrix(testreviews)
dtm.te.idf <- DocumentTermMatrix(testreviews, control=list(weighting = weightTfIdf))

#default weight - test
terms95 <- names(dtm95)[-length(names(dtm95))]
dtm.te95 <- dtm.te[,terms95]
dtm.te95 <- as.data.frame(inspect(dtm.te95))
dtm.te95 <- cbind(dtm.te95, genre_class = c(rep("horror", 92), rep("strategy", 92)))

terms90 <- names(dtm90)[-length(names(dtm90))]
dtm.te90 <- dtm.te[,terms90]
dtm.te90 <- as.data.frame(inspect(dtm.te90))
dtm.te90 <- cbind(dtm.te90, genre_class = c(rep("horror", 92), rep("strategy", 92)))

#tfidf weight - test
dtm.te.idf95 <- dtm.te.idf[,terms95]
dtm.te.idf95 <- as.data.frame(inspect(dtm.te.idf95))
dtm.te.idf95 <- cbind(dtm.te.idf95, genre_class = c(rep("horror", 92), rep("strategy", 92)))

dtm.te.idf90 <- dtm.te.idf[,terms90]
dtm.te.idf90 <- as.data.frame(inspect(dtm.te.idf90))
dtm.te.idf90 <- cbind(dtm.te.idf90, genre_class = c(rep("horror", 92), rep("strategy", 92)))
```

FEATURE SELECTION

```
###INFORMATION GAIN
library(FSelector)
#No sparse condition
dtm.ig <- as.data.frame(inspect(dtm))
dtm.idf.ig <- as.data.frame(inspect(dtm.idf))

#Combined with sparse:
dtm.ig95 <- as.data.frame(inspect(removeSparseTerms(dtm, 0.95)))
dtm.ig90 <- as.data.frame(inspect(removeSparseTerms(dtm, 0.90)))

dtm.idf.ig95 <- as.data.frame(inspect(removeSparseTerms(dtm.idf, 0.95)))
dtm.idf.ig90 <- as.data.frame(inspect(removeSparseTerms(dtm.idf, 0.90)))

#Adding class column:
dtm.ig <- cbind(dtm.ig, genre_class = c(rep("horror", 208), rep("strategy", 208)))
dtm.idf.ig <- cbind(dtm.idf.ig, genre_class = c(rep("horror", 208), rep("strategy", 208)))

dtm.ig95 <- cbind(dtm.ig95, genre_class = c(rep("horror", 208), rep("strategy", 208)))
dtm.ig90 <- cbind(dtm.ig90, genre_class = c(rep("horror", 208), rep("strategy", 208)))

dtm.idf.ig95 <- cbind(dtm.idf.ig95, genre_class = c(rep("horror", 208), rep("strategy", 208)))
dtm.idf.ig90 <- cbind(dtm.idf.ig90, genre_class = c(rep("horror", 208), rep("strategy", 208)))

#Apply information gain:
dtm.ig <- dtm.ig[, c(which(information.gain(genre_class ~., dtm.ig)$attr_importance > 0.03), ncol(dtm.ig))]
```

```
dtm.idf.ig <- dtm.idf.ig[, c(which(information.gain(genre_class ~., dtm.idf.ig)$attr_importance > 0.03),
ncol(dtm.idf.ig))]

dtm.ig95 <- dtm.ig95[, c(which(information.gain(genre_class ~., dtm.ig95)$attr_importance > 0.03),
ncol(dtm.ig95))]
dtm.ig90 <- dtm.ig90[, c(which(information.gain(genre_class ~., dtm.ig90)$attr_importance > 0.03),
ncol(dtm.ig90))]

dtm.idf.ig95 <- dtm.idf.ig95[, c(which(information.gain(genre_class ~., dtm.idf.ig95)$attr_importance >
0.03), ncol(dtm.idf.ig95))]
dtm.idf.ig90 <- dtm.idf.ig90[, c(which(information.gain(genre_class ~., dtm.idf.ig90)$attr_importance >
0.03), ncol(dtm.idf.ig90))]


#INFO GAIN Test sets:
dtm.te <- DocumentTermMatrix(testreviews)
dtm.te.idf <- DocumentTermMatrix(testreviews, control=list(weighting = weightTfIdf))

terms.ig <- names(dtm.ig)[-length(names(dtm.ig))]
dtm.te.ig <- dtm.te[,terms.ig]
dtm.te.ig <- as.data.frame(inspect(dtm.te.ig))
dtm.te.ig <- cbind(dtm.te.ig, genre_class = c(rep("horror", 92), rep("strategy", 92)))

terms.idf.ig <- names(dtm.idf.ig)[-length(names(dtm.idf.ig))]
dtm.te.idf.ig <- dtm.te.idf[,terms.idf.ig]
dtm.te.idf.ig <- as.data.frame(inspect(dtm.te.idf.ig))
dtm.te.idf.ig <- cbind(dtm.te.idf.ig, genre_class = c(rep("horror", 92), rep("strategy", 92)))

terms.ig95 <- names(dtm.ig95)[-length(names(dtm.ig95))]
dtm.te.ig95 <- dtm.te[,terms.ig95]
dtm.te.ig95 <- as.data.frame(inspect(dtm.te.ig95))
dtm.te.ig95 <- cbind(dtm.te.ig95, genre_class = c(rep("horror", 92), rep("strategy", 92)))

terms.idf.ig95 <- names(dtm.idf.ig95)[-length(names(dtm.idf.ig95))]
dtm.te.idf.ig95 <- dtm.te.idf[,terms.idf.ig95]
dtm.te.idf.ig95 <- as.data.frame(inspect(dtm.te.idf.ig95))
dtm.te.idf.ig95 <- cbind(dtm.te.idf.ig95, genre_class = c(rep("horror", 92), rep("strategy", 92)))

terms.ig90 <- names(dtm.ig90)[-length(names(dtm.ig90))]
dtm.te.ig90 <- dtm.te[,terms.ig90]
dtm.te.ig90 <- as.data.frame(inspect(dtm.te.ig90))
dtm.te.ig90 <- cbind(dtm.te.ig90, genre_class = c(rep("horror", 92), rep("strategy", 92)))

terms.idf.ig90 <- names(dtm.idf.ig90)[-length(names(dtm.idf.ig90))]
dtm.te.idf.ig90 <- dtm.te.idf[,terms.idf.ig90]
dtm.te.idf.ig90 <- as.data.frame(inspect(dtm.te.idf.ig90))
dtm.te.idf.ig90 <- cbind(dtm.te.idf.ig90, genre_class = c(rep("horror", 92), rep("strategy", 92)))


#To make it easier to choose the information gain data frame, we created this code and function:
train <-  list(dtm.ig, dtm.ig90, dtm.ig95, dtm.idf.ig, dtm.idf.ig90, dtm.idf.ig95)

inf.gain <- function(train, weight = "tf", min.inf = 0) {
    require(FSelector)
    newig <<- train[, c(which(information.gain(genre_class ~., train)$attr_importance > min.inf),
ncol(train))]
    newigterms <<- names(newig)[-length(names(newig))]
    if(weight == "tf") {
        newig.te <<- dtm.te[,newigterms]
    } else if (weight == "tfidf") {
        newig.te <<- dtm.te.idf[,newigterms]
    }
    newig.te <<- as.data.frame(inspect(newig.te))
    newig.te <<- cbind(newig.te, genre_class = c(rep("horror", 92), rep("strategy", 92)))
}
```

CLASSIFICATION MODELS AND PERFORMANCE EVALUATION

```
#Run model function:
run.model <- function(train, test, method, seed = "5465", return_model = FALSE) {
    if(method == "nb") {
        library(RWeka)
```

```
        set.seed(seed)
        NB <- make_Weka_classifier("weka/classifiers/bayes/NaiveBayes")
        model <- NB(genre_class ~., train)
        cat("\n", "Training set confusion matrix:", sep = "")
        print(table(train[, ncol(train)], predict(model, train[,-ncol(train)])))
        cat("\n", "Test set confusion matrix:", sep = "")
        print(table(test[, ncol(test)], predict(model, test[,-ncol(test)])))
    } else {
        library(caret)
        set.seed(seed)
        model <- train(train[,-ncol(train)], train[, ncol(train)], method = method)
        print(model)
        if(method == "knn") set.seed(seed)
        cat("\n", "Test set confusion matrix:", sep = "")
        print(table(test[, ncol(test)], predict(model, test[,-ncol(test)])))
    }
    if(return_model == TRUE) return(model)
}

#Decision trees, no information gain:
run.model(dtm90, dtm.te90, "rpart")
run.model(dtm95, dtm.te95, "rpart")
run.model(dtm.idf90, dtm.idf.te90, "rpart")
run.model(dtm.idf95, dtm.idf.te95, "rpart")

#KNN:
run.model(dtm90, dtm.te90, "knn")
run.model(dtm95, dtm.te95, "knn")
run.model(dtm.idf90, dtm.idf.te90, "knn")
run.model(dtm.idf95, dtm.idf.te95, "knn")

#Naive Bayes:
run.model(dtm90, dtm.te90, "nb")
run.model(dtm95, dtm.te95, "nb")
run.model(dtm.idf90, dtm.idf.te90, "nb")
run.model(dtm.idf95, dtm.idf.te95, "nb")

#Neural network:
run.model(dtm90, dtm.te90, "nnet")
run.model(dtm95, dtm.te95, "nnet")
run.model(dtm.idf90, dtm.idf.te90, "nnet")
run.model(dtm.idf95, dtm.idf.te95, "nnet")

#SVM radial:
run.model(dtm90, dtm.te90, "svmRadial")
run.model(dtm95, dtm.te95, "svmRadial")
run.model(dtm.idf90, dtm.idf.te90, "svmRadial")
run.model(dtm.idf95, dtm.idf.te95, "svmRadial")


#With inf.gain function and information gain data sets:
inf.gain(train[[1]], "tf", 0.03)
run.model(newig, newig.te, "rpart")
run.model(newig, newig.te, "knn")
run.model(newig, newig.te, "nb")
run.model(newig, newig.te, "nnet")
run.model(newig, newig.te, "svmRadial")

inf.gain(train[[2]], "tf", 0.03)
run.model(newig, newig.te, "rpart")
run.model(newig, newig.te, "knn")
run.model(newig, newig.te, "nb")
run.model(newig, newig.te, "nnet")
run.model(newig, newig.te, "svmRadial")

inf.gain(train[[3]], "tf", 0.03)
run.model(newig, newig.te, "rpart")
run.model(newig, newig.te, "knn")
run.model(newig, newig.te, "nb")
run.model(newig, newig.te, "nnet")
run.model(newig, newig.te, "svmRadial")

inf.gain(train[[4]], "tfidf", 0.03)
run.model(newig, newig.te, "rpart")
```

```
run.model(newig, newig.te, "knn")
run.model(newig, newig.te, "nb")
run.model(newig, newig.te, "nnet")
run.model(newig, newig.te, "svmRadial")

inf.gain(train[[5]], "tfidf", 0.03)
run.model(newig, newig.te, "rpart")
run.model(newig, newig.te, "knn")
run.model(newig, newig.te, "nb")
run.model(newig, newig.te, "nnet")
run.model(newig, newig.te, "svmRadial")

inf.gain(train[[6]], "tfidf", 0.03)
run.model(newig, newig.te, "rpart")
run.model(newig, newig.te, "knn")
run.model(newig, newig.te, "nb")
run.model(newig, newig.te, "nnet")
run.model(newig, newig.te, "svmRadial")

##To find out the misclassified reviews:
#Looking into some of the better performing models:

#NB - idf95
mo.nb.idf95 <- run.model(dtm.idf.95, dtm.idf.te95, "nb")
rownames(dtm.te.idf95[which(dtm.te.idf95$genre_class != predict(mo.nb.idf95, dtm.te.idf95[,-
ncol(dtm.te.idf95)])) , ])
# [1] "Amnesia 28 Snake.txt"              "Amnesia 66 Aridere.txt"            "DreadOut 27 Ragnarokgc.txt"
"DreadOut 50 Commander Kotori.txt"
# [5] "Outlast 48 mostly nice.txt"        "Outlast 53 straightdopeorbit .txt" "Outlast 55 Greta.txt"
"Civ5 74 DalekSupremacy.txt"
# [9] "Endless Legend 63 Philosfr.txt"    "XCOM 4 MrBubbles.txt"              "XCOM 50 Salty.txt"
"XCOM 52 badassgrunt.txt"

#NNET - idf95
mo.nnet.idf95 <- run.model(dtm.idf.95, dtm.idf.te95, "nnet")
rownames(dtm.te.idf95[which(dtm.te.idf95$genre_class != predict(mo.nnet.idf95, dtm.te.idf95[,-
ncol(dtm.te.idf95)])) , ])
# [1] "Amnesia 2 King Spodes.txt"        "DreadOut 50 Commander Kotori.txt" "DreadOut 6 Virtual.txt"
"Outlast 47 Virtual.txt"
# [5] "Outlast 55 Greta.txt"             "Civ5 33 brampage.txt"             "Civ5 40 Clam Bake.txt"
"XCOM 4 MrBubbles.txt"
# [9] "XCOM 50 Salty.txt"

#svmRadial - idf95
mo.svm.idf95<- run.model(dtm.idf.95, dtm.idf.te95, "svmRadial")
rownames(dtm.te.idf95[which(dtm.te.idf95$genre_class != predict(mo.svm.idf95, dtm.te.idf95[,-
ncol(dtm.te.idf95)])), ])
# [1] "Amnesia 2 King Spodes.txt"         "Amnesia 28 Snake.txt"              "Amnesia 65 Daat One Guy.txt"
"DreadOut 11 Lizten.txt"
# [5] "Outlast 47 Virtual.txt"            "Outlast 53 straightdopeorbit .txt" "Outlast 55 Greta.txt"
"Civ5 33 brampage.txt"
# [9] "Civ5 72 Reyals.txt"                "Civ5 74 DalekSupremacy.txt"        "Endless Legend 59
maestro35.txt"   "XCOM 4 MrBubbles.txt"

#NB - idf90
mo.nb.idf90 <- run.model(dtm.idf.90, dtm.idf.te90, "nb")
rownames(dtm.te.idf90[which(dtm.te.idf90$genre_class != predict(mo.nb.idf90, dtm.te.idf90[,-
ncol(dtm.te.idf90)])) , ])
# [1] "Amnesia 10 WarriorVet.txt"         "Amnesia 32 HyDraWx.txt"            "Amnesia 58 jefequeso.txt"
"Amnesia 66 Aridere.txt"
# [5] "DreadOut 11 Lizten.txt"            "Outlast 14 arvix8.txt"             "Outlast 53 straightdopeorbit
.txt" "Outlast 55 Greta.txt"
# [9] "Civ5 40 Clam Bake.txt"             "XCOM 10 Taberone.txt"              "XCOM 4 MrBubbles.txt"
"XCOM 52 badassgrunt.txt"
```