DMDW PROJECT

Ganesh K, Vijayaraghavan V and Kishore P

26/03/2021

KNN ALGORITHM

K-Nearest Neighbor(KNN) Algorithm for Machine Learning 1.K-Nearest Neighbour is one of the simplest Machine Learning algorithms based on Supervised Learning technique.

2.K-NN algorithm assumes the similarity between the new case/data and available cases and put the new case into the category that is most similar to the available categories.

3.K-NN algorithm stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well suite category by using K- NN algorithm.

4.K-NN algorithm can be used for Regression as well as for Classification but mostly it is used for the Classification problems.

5.K-NN is a non-parametric algorithm, which means it does not make any assumption on underlying data.

6.It is also called a lazy learner algorithm because it does not learn from the training set immediately instead it stores the dataset and at the time of classification, it performs an action on the dataset.

7.KNN algorithm at the training phase just stores the dataset and when it gets new data, then it classifies that data into a category that is much similar to the new data.

STEPS

STEP 1: in order for the KNN algorithm to run effectively, it needs a set of libraries to be installed.

```
1.tidyverse (The packages under the tidyverse umbrella help us in performing and interacting with the da
2.class (The knn () function needs to be used to train a model for which we need to install a package 'c
3.caret (The caret package contains functions to streamline the model training process for complex regre
4.gmodels (Various R programming tools for model fitting.)
```

```
#installing and importing required packages
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.0 --
```

```
## v ggplot2 3.3.3     v purrr   0.3.4
## v tibble  3.1.0     v dplyr   1.0.5
## v tidyr   1.1.3     v stringr 1.4.0
## v readr   1.4.0     v forcats 0.5.1
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library(class)
library(caret)
```

```
## Loading required package: lattice
```

```
##
## Attaching package: 'caret'
```

```
## The following object is masked from 'package:purrr':
##
##     lift
```

```
library(gmodels)
```

STEP 2: Import the dataset from the local disk

```
we used the netflix_titles dataset from kaggle to find some kind of patterns and relationships in the da
```

```
#importing the dataset
netflix <- read.csv("D:/PROJECTS/DMDW project/netflix_titles.csv")
```

STEP 3: Here, we are creating a subset ('show_id', 'release_year', 'rating', 'country', 'type') to the original dataset with the data variable named netflix_subset and converting the blank spaces to NA values, so that we can later omit them. We then view the first few values to check if things have turned out how we wanted to be them to be.

```
# creating a subset of our datset with the required predictor variables
netflix.subset <- netflix[c('show_id', 'release_year', 'rating', 'country', 'type')]
netflix.subset[netflix.subset==""] <- NA
```

```
#viewing a part of the dataset to check if it is correct
head(netflix.subset)
```

```
##   show_id release_year rating       country type
## 1       1         2020  TV-MA        Brazil    1
## 2       2         2016  TV-MA        Mexico    0
## 3       3         2011      R     Singapore    0
## 4       4         2009  PG-13 United States    0
## 5       5         2008  PG-13 United States    0
## 6       6         2016  TV-MA        Turkey    1
```

STEP 4: Str method is used to view the structure of the subset to know whether the data inside a variable is an integer or a character or etc .

```
#used to see how the dataset looks
str(netflix.subset)
```

```
## 'data.frame':    7787 obs. of  5 variables:
##  $ show_id     : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ release_year: int  2020 2016 2011 2009 2008 2016 2019 1997 2019 2008 ...
##  $ rating      : chr  "TV-MA" "TV-MA" "R" "PG-13" ...
##  $ country     : chr  "Brazil" "Mexico" "Singapore" "United States" ...
##  $ type        : int  1 0 0 0 0 1 0 0 0 0 ...
```

STEP 5: Here, we are checking whether a specified variable has any null values present in it, if so it will give us the count of the null values. We also need to change the data type of variables with integer to numeric, because integer can only hold whole values while numeric can hold the decimal numbers which can help us to find more accurate values.

```
#checking for incomplete dataset
sum(is.na(netflix.subset$country))
```

```
## [1] 507
```

```
netflix.subset['show_id','release_year'] <- lapply(netflix.subset['show_id','release_year'],as.numeric)
netflix.subset['type'] <- lapply(netflix.subset['type'],as.numeric)
```

STEP 6: We are now removing the null values which were found in the specified variable and we are checking again to know if there are any null values still present.

```
#correcting the incomplete rows
netflix.subset <- na.omit(netflix.subset)
sum(is.na(netflix.subset$country))
```

```
## [1] 0
```

STEP 7: using one-hot encoding [One hot encoding is a process by which categorical variables are converted into a form that could be provided to ML algorithms to do a better job in prediction.] , we are giving a random value for data inside the specified variables. the data to which we did the one-hot encoding process is now available in the 'newdata' dataframe and we are merging it with the original netflix.subset dataframe after dropping the non one hot encoded data in the original subset dataframe.

```
#one-hot encoding the dataframe
dummy <- dummyVars(" ~ .", data=netflix.subset[, c("show_id", "rating", "country")])
newdata <- data.frame(predict(dummy, newdata = netflix.subset[, c("show_id", "rating", "country")]))
```

```
#dropping and merging into final dataframe
drop <- c("country","rating")
netflix.subset = netflix.subset[,!(names(netflix.subset) %in% drop)]
head(netflix.subset)
```

```
##   show_id release_year type
## 1       1         2020    1
## 2       2         2016    0
## 3       3          2011     0
## 4       4          2009     0
## 5       5          2008     0
## 6       6          2016     1
```

```
netflix.subset  = merge(netflix.subset, newdata, by.x = "show_id")
glimpse(netflix.subset)
```

```
## Rows: 7,274
## Columns: 98
## $ show_id                     <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,~
## $ release_year                <dbl> 2020, 2016, 2011, 2009, 2008, 2016, 2019, ~
## $ type                        <dbl> 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, ~
## $ ratingG                     <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
## $ ratingNC.17                 <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
## $ ratingNR                    <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
## $ ratingPG                    <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
## $ ratingPG.13                 <dbl> 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
## $ ratingR                     <dbl> 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, ~
## $ ratingTV.14                 <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, ~
## $ ratingTV.G                  <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
## $ ratingTV.MA                 <dbl> 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, ~
## $ ratingTV.PG                 <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
## $ ratingTV.Y                  <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
## $ ratingTV.Y7                 <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
## $ ratingTV.Y7.FV              <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
## $ ratingUR                    <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
## $ countryArgentina            <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
## $ countryAustralia            <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
## $ countryAustria              <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
## $ countryBangladesh           <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
## $ countryBelarus              <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
## $ countryBelgium              <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
## $ countryBrazil               <dbl> 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
## $ countryBulgaria             <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
## $ countryCambodia             <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
## $ countryCanada               <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
## $ countryChile                <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
## $ countryChina                <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
## $ countryColombia             <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
## $ countryCroatia              <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
## $ countryCyprus               <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
## $ countryCzech.Republic       <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
## $ countryDenmark              <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
## $ countryEgypt                <dbl> 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, ~
## $ countryFinland              <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
## $ countryFrance               <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
## $ countryGeorgia              <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
## $ countryGermany              <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
## $ countryGhana                <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
## $ countryGreece               <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
## $ countryGuatemala            <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
## $ countryHong.Kong            <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
## $ countryHungary              <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
```

STEP 8: We are splitting the available data into training and test parts by assigning 70% of the data to the training set and the remaining 30% to the test set. Now, we are creating a separate label for the "type" variable which is our target variable for finding the accuracy of our model and then we are getting the number of rows available for observation in the it using NROW method.

```
#creating training and testing set
set.seed(123)
dat.d <- sample(1:nrow(netflix.subset),size=nrow(netflix.subset)*0.7,replace = FALSE) #random selection

train.netflix <- netflix.subset[dat.d,] # 70% training data
test.netflix <- netflix.subset[-dat.d,] # remaining 30% test data
```

```
#Creating separate dataframe for 'type' feature which is our target. We give 3 here because that is the
train.netflix_labels <- netflix.subset[dat.d,3]
test.netflix_labels <-netflix.subset[-dat.d,3]
```

STEP 9: We are now training our model with a k of 40. Usually we set k to the square root of the total number of observations but we found that k=40 was better for out model.

```
#Find the number of observation
NROW(train.netflix_labels)
```

```
## [1] 5091
```

```
knn.40 <- knn(train=train.netflix, test=test.netflix, cl=train.netflix_labels, k=40)
```

STEP 10: Now, we are finding the accuracy at k=40 of our trained model by comparing it the orginal labels.

```
#Calculate the proportion of correct classification for k = 40
ACC.40 <- 100 * sum(test.netflix_labels == knn.40)/NROW(test.netflix_labels)
```

STEP 11: We find that our model has an accuracy of ~69% Now we describe the performance of our model with a confusion matrix. A confusion matrix is a table that is often used to describe the performance of a classification model (or "classifier") on a set of test data for which the true values are known. The confusion matrix itself is relatively simple to understand, but the related terminology can be confusing.

```
#creating a confusion matrix
confusionMatrix(table(knn.40 ,test.netflix_labels))
```

```
## Confusion Matrix and Statistics
##
##       test.netflix_labels
## knn.40    0    1
##      0 1507  658
##      1   12    6
##
##                Accuracy : 0.6931
##                  95% CI : (0.6733, 0.7124)
##     No Information Rate : 0.6958
##     P-Value [Acc > NIR] : 0.6199
##
```

```
##                  Kappa : 0.0016
##
##  Mcnemar's Test P-Value : <2e-16
##
##            Sensitivity : 0.992100
##            Specificity : 0.009036
##         Pos Pred Value : 0.696074
##         Neg Pred Value : 0.333333
##             Prevalence : 0.695831
##         Detection Rate : 0.690334
##   Detection Prevalence : 0.991754
##      Balanced Accuracy : 0.500568
##
##       'Positive' Class : 0
##
```

STEP 12: We then use crosstable to get a more clearer understanding of our model. A crosstable is a common type of table featuring a matrix of values between two or more orthogonal lists of header data, of which one is used as column headers.

```
#evaluation with cross table
CrossTable(test.netflix_labels, knn.40)
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## | Chi-square contribution |
## |           N / Row Total |
## |           N / Col Total |
## |         N / Table Total |
## |-------------------------|
##
##
## Total Observations in Table:  2183
##
##
##                     | knn.40
## test.netflix_labels |         0 |         1 | Row Total |
## --------------------|-----------|-----------|-----------|
##                   0 |      1507 |        12 |      1519 |
##                     |     0.000 |     0.022 |           |
##                     |     0.992 |     0.008 |     0.696 |
##                     |     0.696 |     0.667 |           |
##                     |     0.690 |     0.005 |           |
## --------------------|-----------|-----------|-----------|
##                   1 |       658 |         6 |       664 |
##                     |     0.000 |     0.050 |           |
##                     |     0.991 |     0.009 |     0.304 |
##                     |     0.304 |     0.333 |           |
##                     |     0.301 |     0.003 |           |
## --------------------|-----------|-----------|-----------|
##        Column Total |      2165 |        18 |      2183 |
##                     |     0.992 |     0.008 |           |
## --------------------|-----------|-----------|-----------|
##
##
```

- Done by:
  - **Ganesh.K** 212219040033
  - **Vijaya raghavan.V** 212219040177
  - **Kishore.P** 212219040063