Projekt Big Data, Money

Maciej Łosiewicz, Adrian Grabowski, Angelika Włodarczyk, Dawid Bułat
album 256319, 271775, 255685, 256320

10 lipca 2025

Spis treści

W tym sprawozdaniu zajmiemy się analizą danych ze zrzutu z sieci Stack Exchange, a dokładniej z "money.stackexchange.com". Zbudujemy model predykcyjny, sprawdzający czy na dane pytanie została udzielona zaakceptowana odpowiedź, korzystając z danych ze zrzutu. Stworzymy także nowe cechy, a następnie na podstawie analizy spróbujemy wyciągnąć wnioski na temat tego, jaki model się najlepiej dopasował oraz sprawdzimy, czy na podstawie wniosków możemy coś powiedzieć o zachowaniu użytkowników tego portalu.

1 Wprowadzenie danych

W tej krótkiej sekcji dokonamy wprowadzenia danych.

```
sparkR.session(master = "local[*]",
               sparkConfig = list(spark.driver.memory="2g"),
               sparkPackages = "com.databricks:spark-xml_2.12:0.17.0")

## W poniższych funkcjach należy wpisać ścieżkę ze swojego urządzenia

posts <- read.df("E:/1RStudio/Big Data/Projekt/Posts.xml",
                 source = "com.databricks.spark.xml",
                 rootTag = "posts", rowTag = "row")
users <- read.df("E:/1RStudio/Big Data/Projekt/Users.xml",
                 source = "com.databricks.spark.xml",
                 rootTag = "users", rowTag = "row")
```

Zmienimy też nazw kolumny, by były one prostsze do odczytania i użytku.

```
stare_nazwy_posts <- colnames(posts)
nowe_nazwy_posts <- c("AcceptedAnswerId", "AnswerCount", "Body",
                      "ClosedDate", "CommentCount", "CommunityOwnedDate",
                      "ContentLicense", "CreationDate", "FavoriteCount",
                      "PostId", "LastActivityDate", "LastEditDate",
                      "LastEditorDisplayName", "LastEditorUserId",
                      "OwnerDisplayName", "OwnerUserId", "ParentId",
                      "PostTypeId", "Score", "Tags", "Title", "ViewCount")

stare_nazwy_users <-colnames(users)
nowe_nazwy_users <-c("AboutMe", "AccountId", "CreationDate", "DisplayName",
                     "DownVotes", "Id", "LastAccessDate", "Location",
                     "Reputation", "UpVotes", "Views", "WebsiteUrl")

for (i in 1:length(stare_nazwy_posts)) {
  posts <- withColumnRenamed(posts, stare_nazwy_posts[i], nowe_nazwy_posts[i])
}

for (i in 1:length(stare_nazwy_users)) {
  users <- withColumnRenamed(users, stare_nazwy_users[i], nowe_nazwy_users[i])
}
```

2 Przygotowanie danych

Teraz przygotujemy dane do użytku. Zacznijmy od pozbycia się zbędnych nam kolumn.

```
users <- users |> drop(c("AboutMe", "AccountId", "CreationDate", "Location",
                         "WebsiteUrl", "LastAccessDate"))

posts <- withColumn(posts, "BodyLength", NA)
posts$BodyLength <- length(posts$Body)
```

Teraz stworzymy zbiór pytań, a także nowy binarny atrybut mówiący nam, czy jakakolwiek odpowiedź została zatwiedzona.

```
questions <- posts[posts$PostTypeId==1,
                   c("PostId", "BodyLength", "AcceptedAnswerId", "Tags",
                     "AnswerCount", "OwnerUserId")]

questions <- questions |>
  withColumn("AcceptedAnswerId2", questions$AcceptedAnswerId)

# Pozbywamy się wartości NA i podstawiamy 0.

questions <- questions |> fillna(0, cols = "AcceptedAnswerId2")
```

Następnie, dla wartości "0" zapisujemy, że nie było zatwierdzonej odpowiedzi, i analogicznie dla "1" zapisujemy, że została zatwiedzona odpowiedź.

```
questionsWithNA <-
  questions[questions$AcceptedAnswerId2==0] |> withColumn("AnyAccepted", 0)

questionsWithoutNA <-
  questions[questions$AcceptedAnswerId2!=0] |> withColumn("AnyAccepted", 1)
```

Teraz możemy połączyć kolumny po "Id" oraz usunąć poprzednią, pomocniczą kolumnę. Przygotujemy także tagi, by były bez znaków "<", ">", oraz by były oddzielane przecinkiem.

```
questions <- orderBy(union(questionsWithNA, questionsWithoutNA),
                     "PostId") |> drop("AcceptedAnswerId2")

tags_in_array <- questions[,c("PostId", "Tags")] |>
  withColumn("TagsSplit",
             split_string(rtrim(ltrim(questions$Tags, '<'), '>'), '><'))


#head(collect(tags_in_array))
#Możemy sprawdzić, że tagi w rzeczy samej są rozdzielone przecinkiem.
```

Podsumowujemy przygotowanie danych poprzez rozbicie listy na wiele wierszy i poprawienie nazw kolumn.

```
question_tags <- tags_in_array |>
  withColumn('tag', explode(tags_in_array$TagsSplit))
group_of_tags <- question_tags[,c("PostId", "tag")] |>
  groupBy("PostId") |> count()
group_of_tags <- withColumnRenamed(group_of_tags, "count", "TagsCount")
group_of_tags <- withColumnRenamed(group_of_tags, "PostId", "__Id")

questions <- join(questions, group_of_tags,
                  questions$PostId==group_of_tags$`__Id`)
questions <- questions |> drop(col = c("__Id"))
```

3 Analiza danych

W tej części podsumujemy przygotowanie danych i stworzymy modele. Zaczniemy od zliczania ilości pytań względem użytkownika, a także połączenia table "questions" i "users"

```
group_by_user <- questions |> groupBy("OwnerUserId") |>
  count() |> orderBy("count", decreasing = T)
ques_with_users <-
  join(questions, users, questions$OwnerUserId==users$Id) |>
  drop(c("Id", "OwnerUserId"))
```

Rozdzielmy także dane na dane treningowe treningowe i testowe.

```
df_list <- randomSplit(ques_with_users, c(8,2), 100)
train <- df_list[[1]]
test <- df_list[[2]]
```

Zaczniemy od modelu logistycznego.

```
M1 <- spark.logit(train, AnyAccepted ~ BodyLength + AnswerCount + TagsCount +
                    Reputation + Views + UpVotes + DownVotes)
predictions_M1 <- predict(M1, test)[,c("AnyAccepted", "prediction")]
summary(M1)

## $coefficients
##                   Estimate
## (Intercept) -6.385346e-01
## BodyLength   1.755551e-06
## AnswerCount  1.669334e-01
## TagsCount    2.904360e-02
## Reputation   1.527266e-05
## Views        9.709913e-05
## UpVotes      2.291229e-04
## DownVotes   -1.035584e-03
```

Jak można zauważyć, "Views", "Reputation" oraz "BodyLength" mają wpływ rzędu 10^{-5}, więc stworzymy model który nie bierze ich pod uwagę.

```
M1 <- spark.logit(train, AnyAccepted ~ AnswerCount + TagsCount +
                      UpVotes + DownVotes)
predictions_M1 <- predict(M1, test)[,c("AnyAccepted", "prediction")]
summary(M1)

## $coefficients
##                   Estimate
## (Intercept) -0.6184378241
## AnswerCount  0.1682073310
## TagsCount    0.0302422041
## UpVotes      0.0003364579
## DownVotes   -0.0004391022
```

Jak widzimy, zmienne nie podległy drastycznym zmianom, więc możemy założyć że zmienne których się pozbyliśmy miały w rzeczy samej marginalny wpływ na akceptacje odpowiedzi.

Teraz stwórzmy funkcję która wyliczy nam odpowiednie miary.

```
measures <- function(predictions, p=0.5){
  d1 <- predictions[predictions$AnyAccepted==1]
  FN <- count(d1[d1$prediction<p])  # False Negative
  TP <- count(d1[d1$prediction>=p]) # True Positive

  d2 <- predictions[predictions$AnyAccepted==0]
  TN <- count(d2[d2$prediction<p])  # True Negative
  FP <- count(d2[d2$prediction>=p]) # False Positive

  ACC <- (TP+TN)/(TP+FN+TN+FP)      # Dokładność
  FDR <- FP/(TP+FP)                 # False discovery rate (1-precyzja)
  TPR <- TP/(TP+FN)                 # Czułość
  TNR <- TN/(FP+TN)                 # Swoistość
  result <- matrix(c(TP,FN,TN,FP,ACC,FDR,TPR,TNR), nrow = 1)
  colnames(result) <- c("TP", "FN", "TN", "FP", "ACC", "FDR", "TPR", "TNR")
  return(result)
}

measures(predictions_M1)

##       TP   FN   TN  FP       ACC       FDR       TPR       TNR
## [1,] 768 2577 3527 507 0.5820572 0.3976471 0.2295964 0.8743183
```

Następnie stworzymy modele dla regresji liniowej, lasu losowego i drzewa decyzyjnego.

```
M2 <- spark.lm(train, AnyAccepted ~ AnswerCount + TagsCount +
                      UpVotes + DownVotes)
predictions_M2 <- predict(M2, test)[,c("AnyAccepted", "prediction")]
measures(predictions_M2)

##       TP   FN   TN  FP       ACC       FDR       TPR       TNR
## [1,] 679 2666 3610 424 0.5812441 0.3844062 0.2029895 0.8948934
```

Jak widzimy, model regresji liniowej jest trochę gorszy od modelu logistycznego jeżeli patrzymy na czułość, dokładność i kontrolę fałszywych wyników, ale jest też trochę lepszy jeżeli chodzi o swoistość.

Następnie sprawdźmy model "Random Forest". Algorytm lasu losowego wykorzystuje wiele drzew decyzyjnych, które są budowane na losowych podzbiorach danych treningowych i cech, a następnie kombinuje wyniki tych drzew poprzez głosowanie większościowe, aby uzyskać ostateczną klasyfikację lub prognozę. Dzięki temu zapewnia on wysoką wydajność i odporność na przeuczenie.

```
M3 <- spark.randomForest(train, type = "classification", numTrees = 10,
                  AnyAccepted ~ AnswerCount + TagsCount + UpVotes + DownVotes)
predictions_M3 <- predict(M3, test)[,c("AnyAccepted", "prediction")]
measures(predictions_M3)

##         TP  FN   TN   FP       ACC       FDR       TPR       TNR
## [1,] 2778 567 2424 1610 0.7049736 0.3669098 0.8304933 0.6008924
```

Następnie wykorzystamy model drzewa decyzyjnego. Drzewa decyzyjne są strukturami modelowania danych, które operują na zasadzie hierarchicznych drzew. Rozpoczynając od korzenia, reprezentującego pełen zbiór danych, drzewa decyzyjne dokonują serii podziałów, wybierając kolejne cechy, które najlepiej segregują dane na bardziej jednorodne grupy. Każdy węzeł w drzewie reprezentuje decyzję na podstawie konkretnych cech, prowadząc do kolejnych gałęzi lub liści, które reprezentują ostateczne klasyfikacje lub prognozy. Proces ten jest realizowany poprzez iteracyjne wybieranie najlepszych podziałów na podstawie kryteriów czystości zbioru danych, takich jak entropia czy impurity. Dzięki swojej intuicyjnej strukturze, drzewa decyzyjne są często stosowane do analizy i klasyfikacji danych, jednak wymagają odpowiedniej optymalizacji i przycinania, aby uniknąć zjawiska nadmiernego dopasowania do danych treningowych.

```
M4 <- spark.decisionTree(train, AnyAccepted ~ AnswerCount + TagsCount +
                     UpVotes + DownVotes)
predictions_M4 <- predict(M4, test)[,c("AnyAccepted", "prediction")]
measures(predictions_M4)

##         TP  FN   TN   FP       ACC      FDR       TPR       TNR
## [1,] 2743 602 2460 1574 0.7051091 0.364605 0.8200299 0.6098166
```

Jak widzimy, wyniki obydwu tych modeli są do siebie zbliżone, co pokazuje nam, że wariancja danych jest mała.

	ACC	FDR	TPR	TNR
Model logistyczny	0.5821	0.3976	0.2296	0.8743
Model regresji liniowej	0.5812	0.3844	0.2030	0.8949
Model lasu losowego	0.7050	0.3669	0.8305	0.6009
Model drzewa decyzyjnego	0.7051	0.3646	0.8200	0.6098

Tabela 1: Wartości miar dla różnych modeli predykcyjnych

Podsumowując, wszystkie cztery modele wykorzystane do predykcji danych mają swoje zalety i wady. Jeżeli chcemy model z bardzo wysoką swoistością, powinniśmy użyć modelu

logistycznego bądź regresji liniowej, natomiast jeżeli chcemy model z lepszą dokładnością i kontrolą fałszywych wyników, możemy skorzystać z drzewa decyzyjnego dla mniejszych rozmiarów danych, bądź lasu losowego dla większego rozmiaru danych.

Jak możemy też zobaczyć z wcześniejszej analizy zmiennych, największy wpływ na zatwierdzenie odpowiedzi ma ilość tychże odpowiedzi, co jest spodziewanym wynikiem. Jak możemy też zauważyć, wpływ mają "UpVote'y" oraz "DownVote'y", przy czym oczywiście "DownVote'y" mają negatywny wpływ.