Sprawozdanie z projektu Big Data

Michał Turek, 246993, Michał Maj 256556, Damian Lewańczyk 242999

10 lutego 2024

Spis treści

1 Wstęp

W erze cyfrowej, fora internetowe są skarbnicą wiedzy i doświadczeń użytkowników z całego świata. Jednym z takich miejsc, gdzie gracze mogą dzielić się swoimi przemyśleniami, pytaniami i rozwiązaniami, jest platforma Gaming Stack Exchange. Jest to część szerszej sieci Stack Exchange, która oferuje specjalistyczne fora dla różnych dziedzin i zainteresowań. Forum gaming.stackexchange skupia się na pytaniach i odpowiedziach dotyczących szeroko pojętych gier – od konsolowych po komputerowe, od strategii po gry zręcznościowe.

Celem naszej analizy będzie zrozumienie dynamiki i zachowań użytkowników na tym forum, a w szczególności identyfikacja czynników, które mogą wpływać na otrzymanie zaakceptowanej odpowiedzi na zadane pytanie. Zrozumienie tych mechanizmów może nie tylko rzucić światło na to, jakie treści są najbardziej wartościowe dla społeczności, ale także może pomóc w poprawie jakości interakcji na forum i zwiększeniu szans użytkowników na uzyskanie satysfakcjonujących odpowiedzi.

Do analizy wykorzystamy zbiór danych zebranych ze zrzutów bazy danych naszego forum. Dzielą się one na 8 tabel: Badges, Comments, PostHistory, PostLinks, Posts, Tags, Users i Votes. W naszych badaniach skupimy się głównie na danych zawartych w Posts, Users i Tags. Na ich podstawie wybierzemy zmienne, które najlepiej oddadzą istotę problemu, oraz dobrze wpasują

się do modeli. Stworzymy również kilka własnych zmiennych, na podstawie istniejących, w celu polepszenia dokładności modelu. W celu modelowania wykorzystamy metody regresji logistycznej, drzewa decyzyjnego i lasów losowych, zaimplementowanych w pakiecie *SparkR*. Większość przekształceń na naszych danych wykonamy również wykorzystując Sparkowe funkcje.

2 Opisowa analiza danych

Zacznijmy od opisowej analizy danych w celu lepszego zapoznania się z nimi i ich zrozumieniu, co będzie niezbędne w dalszej części analiz. Tak jak wspomnieliśmy we wstępie, nasze dane dzielą się na 8 tabel. Opiszemy teraz krótko każdą z nich.

2.1 Tabela Posts

Tabela Posts jest rdzeniem forum, zawiera bowiem pytania i odpowiedzi użytkowników. Do jej najważniejszych kolumn należą:

- **Id**: unikalny identyfikator postu.
- **PostTypeId**: identyfikator typu postu, gdzie 1 oznacza pytanie, a 2 odpowiedź. Pozostałe identyfikatory $(3-8)$ nas nie będą interesowały.
- **AcceptedAnswerId**: identyfikator zaakceptowanej odpowiedzi na pytanie. Jest to jedna z najważniejszych dla nas kolumn, ponieważ to na jej podstawie będziemy sprawdzali, czy na dane pytanie dostaliśmy zaakceptowaną odpowiedź.
- **CreationDate**: data utworzenia postu.
- **Score**: punktacja postu (lajki-łapki w dół).
- **ViewCount**: liczba wyświetleń postu (dotyczy tylko pytań).
- **Body**: treść postu w formacie HTML.
- **OwnerUserId**: identyfikator użytkownika, który jest autorem postu. Kolejna ważna zmienna, na podstawie której będziemy łączyli tabele Users i Posts w celu uzyskania informacji o osobie zadającej pytanie oraz odpowiadających na pytania.
- **LastEditorUserId**, **LastEditDate**: informacje o ostatniej edycji postu.
- **Title**: tytuł pytania (dla odpowiedzi pole to jest puste).
- **Tags**: tagi przypisane do pytania.
- **AnswerCount**, **CommentCount**: liczba odpowiedzi i komentarzy do pytania. Zmienne, które również mogą przydać się do naszych modeli.

Tabela zawiera 271948 wierszy, z czego 100470 to posty. Około 64 procent postów ma udzieloną zaakceptowaną odpowiedź. Liczba ta będzie dla nas ważnym punktem odniesienia, ponieważ modelując zmienną binarną określającą, czy na dane pytanie została udzielona zaakceptowana odpowiedź, możemy przypisać z góry same wartości 1, otrzymując 64 procent dokładności. Będziemy oczywiście chcieli uzyskać jak najwięcej ponad ten wynik. Poza wymienionymi kolumnami, w tabeli mamy jeszcze zmienne takie, jak FavoriteCount, ClosedDate (tylko wtedy, kiedy post został zamknięty), CommunityOwnedDate i ContentLicense.

2.2 Tabela Users

Kolejną bardzo ważną tabelą jest Users. Zawiera ona kluczowe informacje o użytkownikach forum, takie jak

- **Id**: unikalny identyfikator użytkownika. Ważny do łączenia tabeli z Posts.
- **Reputation**: punkty reputacji użytkownika, które odzwierciedlają, jak społeczność ocenia wkład danego użytkownika. Jedna z ważniejszych zmiennych, którą będziemy wykorzystywać do modelowania.
- **CreationDate**: data utworzenia konta.
- **DisplayName**: wyświetlana nazwa użytkownika.
- **LastAccessDate**: data ostatniej aktywności użytkownika na forum.
- **WebsiteUrl**, **Location**: opcjonalne pola z adresem strony internetowej i lokalizacją użytkownika.
- **AboutMe**: krótki opis profilu użytkownika.
- **Views**: liczba wyświetleń profilu, wykorzystana w dalszej części analiz do modelowania.
- **UpVotes**, **DownVotes**: łapki w górę i w dół.

W tabeli mamy 209507 wierszy, które odpowiadają liczbie użytkowników forum. Tabela ta bardzo ważna w kontekście modelowania, w którym wykorzystamy zmienne utworzone na podstawie połączenia pól Users (na przykład Reputation, czy Views) z tabelą Posts po zmiennej ID. Poza wymienionymi zmiennymi w tabeli mamy jeszcze ProfileImageUrl - Url zdjęcia profilowego EmailHash (obecnie nie zawiera już danych) oraz AccountId (nie mylić z ID).

2.3 Tabela Tags

Tabela Tags zawiera informacje o tagach używanych do kategoryzacji pytań. Kluczowe pola to:

- **Id**: Unikalny identyfikator tagu.
- **TagName**: Nazwa tagu, która jest używana do kategoryzowania pytań. Będziemy używali tej kolumny do łączenia tabel Posts i Tags.
- **Count**: Liczba postów przypisanych do tego tagu, co może wskazywać na popularność lub znaczenie danego tematu.
- **ExcerptPostId**: Identyfikator postu, który zawiera krótki opis tagu. Jest to użyteczne dla użytkowników, którzy chcą zrozumieć, do czego dany tag służy.
- **WikiPostId**: Identyfikator postu, który zawiera rozszerzony opis tagu w formie wiki. To pozwala społeczności na tworzenie i utrzymywanie bogatych, szczegółowych opisów tagów.

W tabeli mamy 6187 wierszy, a co za tym idzie tagów. Pokazuje to bardzo dużą różnorodność tagów występujących na forum. Ważną dla nas kolumną z tej tabeli jest Count, która posłuży nam jako jedna ze zmiennych do modelu.

2.4 Pozostałe tabele

- **Comments**: zawiera komentarze do postów. Ważne pola to m.in. Id, PostId, Score, Text, CreationDate, UserId.
- **PostHistory**: rejestruje historię zmian w postach, np. edycje, zmiany tagów. Kluczowe kolumny to Id, PostHistoryTypeId, PostId, CreationDate, UserId, Text.
- **PostLinks**: zawiera informacje o powiązaniach między postami, np. duplikaty. Ważne kolumny to Id, CreationDate, PostId, RelatedPostId, LinkTypeId.
- **Votes**: rejestruje głosy na posty i komentarze. Ważne pola to Id, PostId, VoteTypeId, CreationDate.
- **Badges**: reprezentuje odznaki przyznawane użytkownikom. Odznaki są formą uznania dla użytkowników za ich wkład w społeczność. Kluczowe kolumny to: ID, UserID, Name, Date i Class.

Z powyższych tabel nie będziemy już korzystali w dalszych częściach analizy.

3 Wgranie danych

```
# ścieżki
# MT:  C:/Users/micha/OneDrive/Pulpit/big_data_projekt
# DL:  C:/STUDIA/Matematyka ( II stopień )/III semestr/Algorytmy big data/
#Projekt ABD/gaming.stackexchange.com
# MM:  H:/Gaming
#Wczytujemy trzy pliki (SparkR)

df_gaming_posts <- read.df("H:/Gaming/Posts.xml", source =
"com.databricks.spark.xml", rootTag = "posts", rowTag = "row")

df_gaming_users <- read.df("H:/Gaming/Users.xml", source =
"com.databricks.spark.xml", rootTag = "users", rowTag = "row")

df_gaming_tags <- read.df("H:/Gaming/Tags.xml", source =
"com.databricks.spark.xml", rootTag = "tags", rowTag = "row")

df_gaming_badges <- read.df("H:/Gaming/Badges.xml", source =
"com.databricks.spark.xml", rootTag = "badges", rowTag = "row")

df_gaming_comments <- read.df("H:/Gaming/Comments.xml",
source = "com.databricks.spark.xml", rootTag = "comments", rowTag = "row")

df_gaming_posthistory <- read.df("H:/Gaming/Posthistory.xml",
 source = "com.databricks.spark.xml", rootTag = "posthistory", rowTag = "row")

df_gaming_postlinks <- read.df("H:/Gaming/Postlinks.xml",
```

```
source = "com.databricks.spark.xml", rootTag = "postlinks", rowTag = "row")

df_gaming_votes <- read.df("H:/Gaming/Votes.xml", source =
"com.databricks.spark.xml", rootTag = "votes", rowTag = "row")
```

4 Transformacje danych

Tabela *Posts*

```
##     _Id _AcceptedAnswerId _AnswerCount _CommentCount _OwnerUserId _ParentId
## 33  71               195            2             0           16         NA
## 34  73               102            1             0           26         NA
## 35  75               159            2             1           46         NA
## 36  79              1653            1             0           29         NA
## 37  80                NA           NA             6           10         64
## 38  84              2966            1             2           72         NA
## 39  87                98            1             1           46         NA
## 40  90               118            6             0           89         NA
## 41  93                NA           NA             0           58         62
## 42  97              2381            3             1           39         NA
##     _PostTypeId _Score                          _Tags _ViewCount
## 33            1     11                    <xbox-360>       2538
## 34            1     12                    <xbox-360>       2191
## 35            1     14 <nintendo-ds><game-boy-advance>       9519
## 36            1     13          <super-mario-bros-2>      15224
## 37            2     22                          <NA>         NA
## 38            1      8             <everquest><aion>        412
## 39            1     32               <portal-series>       2715
## 40            1     23        <red-dead-redemption>      19811
## 41            2      6                          <NA>         NA
## 42            1      9    <lord-of-the-rings-online>       1218
```

Tabela *Users*:

Tabela 1: 10 wartości z tabeli *Users*

	_UserId	_Reputation	_Views
33	34	203	8
34	35	1088	72
35	36	434	23
36	37	103	8
37	38	936	47
38	39	5567	335
39	40	291	109
40	41	413	41
41	42	131	5
42	43	782	69

Tabela *Tags*:

Tabela 2: 10 wartości z tabeli *Tags*

	_Id	_TagName	_Count
33	133	health	23
34	134	psn	436
35	137	borderlands	176
36	138	ps2	115
37	145	game-boy	33
38	146	nintendo-wii	283
39	147	mafia-wars	8
40	148	facebook	18
41	150	left-4-dead	20
42	151	source-engine	94

W pierwszym kroku dzielimy dane z tabeli Posts na pytania i odpowiedzi. Zmienna, która pomoże nam dokonać tego podziału, to `PostTypeId`. Wiemy, że `PostTypeId`= **1** oznacza pytanie, a `PostTypeId`= **2** odpowiedź na jakieś pytanie. Poniżej przedstawiony jest kod który zwraca takie dwa zbiory danych:

```
df_gaming_questions <- df_gaming_posts[df_gaming_posts$`_PostTypeId`==1, ] |>
  withColumnRenamed("_Id", "_QuestionId") |> withColumnRenamed("_OwnerUserId",
                                                    "_QuestionOwnerUserId")
df_gaming_answers <- df_gaming_posts[df_gaming_posts$`_PostTypeId`==2, ] |>
  withColumnRenamed("_Id", "_AnswerId") |> withColumnRenamed("_OwnerUserId",
                                                    "_AnswerOwnerUserId")
```

Dodanie kolumny `IsAcceptedAnswer` poniżej:

```
df_gaming_questions <- df_gaming_questions |> withColumn("_IsAcceptedAnswer",
ifelse(isNull(df_gaming_questions$`_AcceptedAnswerId`),0,1)) |>
withColumnRenamed("_OwnerUserId", "_QuestionOwnerUserId")
#df_gaming_questions_limit <- df_gaming_questions |> limit (1000)
#View(collect(df_gaming_questions_limi
```

Do obu tabel: pytań i odpowiedzi dołączamy tabelę *Users*. Poniżej kod prezentujący takie łączenia. Wykorzystaliśmy funkcję `join()` z pakieru **SparkR**.

```
df_gaming_questions_with_users <- join(df_gaming_questions,
df_gaming_users,df_gaming_questions$`_QuestionOwnerUserId`==
df_gaming_users$`_UserId`) #[, c("_AnswerId", "_DisplayName")]

# informacje o odpowiedziach wraz z użytkownikami
df_gaming_answers_with_users <-  join(df_gaming_answers,
df_gaming_users,df_gaming_answers$`_AnswerOwnerUserId`==
df_gaming_users$`_UserId`) #[, c("_AnswerId", "_DisplayName")]
```

Poniżej, zamieniamy kilka tagów na jedną listę

```
# usuwamy pierwszy ("<") i ostatni (">") znak z napisu, dzielimy na napisie
#"><" na tablicę tagów
df_gaming_tags_in_array <- df_gaming_questions_with_users|> withColumn("_Tags",
split_string(rtrim(ltrim(df_gaming_questions_with_users$`_Tags`, '<'), '>'),
             '><'))
df_gaming_tags_in_array_limit <- df_gaming_tags_in_array |> limit (1000)
View(collect(df_gaming_tags_in_array_limit))
```

Poniżej rozbijamy listy tagów na wiele wierszy

```
# rozbijamy listę na wiele wierszy, upiększamy nazwy kolumn
df_gaming_questions_to_tags <- df_gaming_tags_in_array |>
withColumn('_tag', explode(df_gaming_tags_in_array$`_Tags`))
#|> SparkR::select(c("user", "tag"))
df_gaming_questions_to_tags <- df_gaming_questions_to_tags[,
c("_QuestionId", "_AcceptedAnswerId", "_QuestionOwnerUserId","_tag")]
```

Poniżej fragment kodu liczący dla każdego pytania średnią liczbę odsłon (Views) i reputacji użytkowników, którzy na nie odpowiedzieli.

```
# liczymy średnią reputacji użytkowników, którzy odpowiedzieli na dany post
#(grupowanie po Id pytania)
df_gaming_avg_rep_views_users_per_question <-
agg(groupBy(df_gaming_answers_with_users,"_ParentId"),
avg(df_gaming_answers_with_users$`_Reputation`),
avg(df_gaming_answers_with_users$`_Views`))
df_gaming_avg_rep_views_users_per_question <-
df_gaming_avg_rep_views_users_per_question |>
withColumnRenamed("avg(_Reputation)", "avg_Reputation_Commenting") |>
withColumnRenamed("avg(_Views)", "avg_Views_Commenting")  |>
withColumnRenamed("_ParentId", "ParentId_User")
```

Następnie dodaliśmy połączyliśmy stworzoną przed chwilą tabelę z tabelą z pytaniami.

```
# Dodajemy tabele wyzej do tabeli Questions

df_gaming_questions_with_agg_rep_views_users <- join(df_gaming_questions,
                              df_gaming_avg_rep_views_users_per_question,

                              df_gaming_questions$`_QuestionId` ==
df_gaming_avg_rep_views_users_per_question$`ParentId_User`)

#df_gaming_questions_with_agg_rep_views_users_limit <-
#df_gaming_questions_with_agg_rep_views_users |> limit(1000)
#View(collect(df_gaming_questions_with_agg_rep_views_users))
```

Ponadto dodajemy Views i Reputation pytającego do naszej tabeli. Poniżej fragment kodu

```
## DODAWANIE REPUTACJI PYTAJACEGO

# dodajemy reputation, views od pytającego, łączymy tabele wyzej z tabelą
#Users, klucz: _QuestionOwnerUserId = _UserId
df_gaming_questions_with_agg_rep_views_users_and_owner_rep <-
join(df_gaming_questions_with_agg_rep_views_users, df_gaming_users,

df_gaming_questions_with_agg_rep_views_users$`_QuestionOwnerUserId` ==
df_gaming_users$`_UserId`)

#df_gaming_questions_with_agg_rep_views_users_and_owner_rep_limit <-
#df_gaming_questions_with_agg_rep_views_users_and_owner_rep |> limit(1000)
#View(collect(df_gaming_questions_with_agg_rep_views_users_and_owner_rep))
```

Dodawanie tagów. Chcemy obliczyć..

Na początek łaczymy tabelę gdzie mamy id pytania i tagi (podzielone na wiersze funkcją explode) z tabelą tags, po nazwie tagu, zliczamy liczbę pytań dla każdego tagu (ile razy występował).

```
#łaczymy tabelę gdzie mamy id pytania i tagi (podzielone na wiersze funkcją
#explode) z tabelą tags, po nazwie tagu, dodając counta dla każdego tagu
df_gaming_questions_tags_counts <- join(df_gaming_questions_to_tags,
                                        df_gaming_tags,
                                        df_gaming_questions_to_tags$`_tag` ==
                                        df_gaming_tags$`_TagName`)
df_gaming_questions_tags_counts <- df_gaming_questions_tags_counts[,
                                   c("_QuestionId",  "_tag", "_Count")]
```

Poniżej zliczamy średnie wystąpień tagów będących w pytaniu

```
# obliczamy średnią ze zmiennej _Count dla tagów z danego pytania

df_gaming_questions_tags_counts_avg <- agg(groupBy(df_gaming_questions_tags_counts,
                                           "_QuestionId"),

                                       avg(df_gaming_questions_tags_counts$`_Count`))

df_gaming_questions_tags_counts_avg <- df_gaming_questions_tags_counts_avg |>
withColumnRenamed("avg(_Count)", "avg_Tag_Count")
df_gaming_questions_tags_counts_avg <- df_gaming_questions_tags_counts_avg |>
withColumnRenamed("_QuestionId", "_QuestionId_v2")
```

Dołączamy tabelę z popularnością tagów do reszty

```
# dołączamy tabelę wyżej do tabeli
#df_gaming_questions_with_agg_rep_views_users_and_owner_rep_limit
#z resztą agregatów (po komentujących i QuestionOwner)
```

```
df_gaming_questions_agg_Commenting_Owner_Tag <-
join(df_gaming_questions_with_agg_rep_views_users_and_owner_rep,
df_gaming_questions_tags_counts_avg,
df_gaming_questions_with_agg_rep_views_users_and_owner_rep$`_QuestionId` ==
df_gaming_questions_tags_counts_avg$`_QuestionId_v2`)

##df_gaming_questions_agg_Commenting_Owner_Tag_limit <-
#df_gaming_questions_agg_Commenting_Owner_Tag |> limit(1000)
#View(collect(df_gaming_questions_agg_Commenting_Owner_Tag_limit))
#View(head(df_gaming_questions_agg_Commenting_Owner_Tag, 20))


#df_gaming_questions_agg_Commenting_Owner_Tag <-
#df_gaming_questions_agg_Commenting_Owner_Tag |>
#withColumnRenamed("_QuestionId.1", "_QuestionId_v2")
```

Wybieramy zmienne, na podstawie których chcemy tworzyć modele:

```
df_gaming_for_model <- df_gaming_questions_agg_Commenting_Owner_Tag |>
withColumnRenamed("_IsAcceptedAnswer","IsAcceptedAnswer") |>
withColumnRenamed("_AnswerCount","AnswerCount") |>
withColumnRenamed("_CommentCount","CommentCount") |>
withColumnRenamed("_Score","Score") |>
withColumnRenamed("_ViewCount","ViewCount") |>
withColumnRenamed("_Reputation", "Reputation_Question_Owner") |>
withColumnRenamed("_Views", "Views_Question_Owner")


df_gaming_for_model <- df_gaming_for_model[, c("IsAcceptedAnswer", "AnswerCount",
                                               "CommentCount",
                                               "Score", "ViewCount",
                                               "avg_Reputation_Commenting",
                                               "avg_Views_Commenting",
                                               "Reputation_Question_Owner",
                                      "Views_Question_Owner", "avg_Tag_Count")]
#colnames(df_gaming_questions_agg_Commenting_Owner_Tag)
#colnames(df_gaming_for_model)
```

```
# STANDARYZACJA ZMIENNYCH  "ViewCount" ,"avg_Reputation_Commenting"
#"avg_Views_Commenting" "Reputation_Question_Owner" "Views_Question_Owner"
#"avg_Tag_Count"


#funkcja do zliczania średniej i odchylenia standardowego
summary <- collect(SparkR::summary(df_gaming_for_model))
```

```r
mean <- summary[2,6]
sd <- summary[3,6]

df_gaming_for_model_standarized <- df_gaming_for_model |>
mutate(ViewCount=(df_gaming_for_model$ViewCount - mean)/sd)
#View(collect(d1))

###### avg_Reputation_Commenting

mean<-summary[2,7]
sd<-summary[3,7]

df_gaming_for_model_standarized <- df_gaming_for_model_standarized |>
mutate(avg_Reputation_Commenting=(df_gaming_for_model$avg_Reputation_Commenting
                                  - mean)/sd)


###### avg_Views_Commenting

mean<-summary[2,8]
sd<-summary[3,8]

df_gaming_for_model_standarized <- df_gaming_for_model_standarized |>
mutate(avg_Views_Commenting=(df_gaming_for_model$avg_Views_Commenting - mean)/sd)


###### Reputation_Question_Owner

mean<-summary[2,9]
sd<-summary[3,9]

df_gaming_for_model_standarized <- df_gaming_for_model_standarized |>
mutate(Reputation_Question_Owner=(df_gaming_for_model$Reputation_Question_Owner
                                  - mean)/sd)


###### Views_Question_Owner

mean<-summary[2,10]
sd<-summary[3,10]

df_gaming_for_model_standarized <- df_gaming_for_model_standarized |>
mutate(Views_Question_Owner=(df_gaming_for_model$Views_Question_Owner - mean)/sd)


###### avg_Tag_count

mean<-summary[2,11]
```

```
sd<-summary[3,11]

df_gaming_for_model_standarized <- df_gaming_for_model_standarized |>
mutate(avg_Tag_Count=(df_gaming_for_model$avg_Tag_Count - mean)/sd)


#df_gaming_for_model_standarized_limit <- df_gaming_for_model_standarized |>
#limit(10)
#View(collect(df_gaming_for_model_standarized_limit))
```

```
df_gaming_for_model_standarized_df_r <- head(df_gaming_for_model_standarized, 10)
df_gaming_for_model_standarized_df_r
```

```
##    IsAcceptedAnswer AnswerCount CommentCount Score   ViewCount
## 1                 0           2            0     0 -0.25573405
## 2                 1           1            0     7 -0.23181934
## 3                 1           2            0     4 -0.25090445
## 4                 1           7            4     7  0.07540885
## 5                 1           3            3     2 -0.15552559
## 6                 0           2            1     4 -0.23765220
## 7                 1           4            0     3 -0.26177689
## 8                 0           3            0     4  0.34908645
## 9                 0           8            1     4 -0.17629055
## 10                0           3            2    -4 -0.20361165
##    avg_Reputation_Commenting avg_Views_Commenting Reputation_Question_Owner
## 1                -0.57715500          -0.53291602               -0.44845871
## 2                 0.69754710          -0.11937631               -0.01999403
## 3                 0.07985966          -0.32010390               -0.37804073
## 4                -0.45434644          -0.34172150               -0.12088746
## 5                 0.15589084          -0.06150089                0.73662695
## 6                -0.57508065          -0.52918403               -0.20029384
## 7                 0.73007385           0.14786062               -0.36027668
## 8                -0.53242449          -0.50679211                0.23492544
## 9                -0.54953079          -0.50619233               -0.44750137
## 10               -0.52067693          -0.36278480               -0.44877783
##    Views_Question_Owner avg_Tag_Count
## 1           -0.39342603    0.04054744
## 2            0.03673676   -0.57235074
## 3           -0.36560793    0.18242734
## 4            0.10164566    0.18242734
## 5            1.61173868    0.18242734
## 6           -0.23081454   -0.56229853
## 7           -0.32625550   -0.33353873
## 8           -0.20186563    0.18242734
## 9           -0.39410452    0.04054744
## 10          -0.39410452   -0.36556223
```

5 testy

Poniżej, dodajemy do pytań, gdzie mieliśmy zaakceptowaną odpowiedź ID użytkownika, który tej odpowiedzi udzielił.

```
df_gaming_questions_tags_users_answers <- join(df_gaming_questions_to_tags,
                    df_gaming_answers[, c("_AnswerId", "_AnswerOwnerUserId")],

df_gaming_questions_to_tags$`_AcceptedAnswerId` == df_gaming_answers$`_AnswerId`)
#df_gaming_questions_tags_users_answers_limit <-
#df_gaming_questions_tags_users_answers |>
#limit (1000)
#View(collect(df_gaming_questions_tags_users_answers_limit))
```

6 Tworzenie modeli

Na początku stworzymy model za pomocą regresji logistycznej, używając funkcji `spark.logit`. Zbadamy przy tym:

- Dokładność (*Accuracy*): Miara, która określa stosunek poprawnie sklasyfikowanych przypadków do ogólnej liczby przypadków w zbiorze danych. Formuła dokładności to:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

- Czułość (*Sensitivity*, *True Positive Rate*): Miara, która określa zdolność modelu do poprawnego zidentyfikowania pozytywnych przypadków. Formuła czułości to:

$$Sensitivity = \frac{TP}{TP + FN}$$

- Swoistość (*Specificity*, *True Negative Rate*): Miara, która określa zdolność modelu do poprawnego zidentyfikowania negatywnych przypadków. Formuła swoistości to:

$$Specificity = \frac{TN}{TN + FP}$$

- Precyzja (*Precision*, *Positive Predictive Value*): Miara, która określa stosunek poprawnie sklasyfikowanych pozytywnych przypadków do wszystkich przypadków sklasyfikowanych jako pozytywne. Formuła precyzji to:

$$Precision = \frac{TP}{TP + FP}$$

- Indeks Youdena (*Youden's J statistic*): Statystyka, która mierzy zdolność testu do poprawnego identyfikowania zarówno pozytywnych, jak i negatywnych przypadków. Indeks Youdena to suma czułości i swoistości minus jeden:

$$Youden = Sensitivity + Specificity - 1$$

Wartości TP,TN,FP i FN są brane z poniższej macierzy pomyłek.

		Predicted	
		Yes	No
Actual	Yes	TP	FN
	No	FP	TN

Wyniki przedstawimy w tabelkach.

```
training <- df_gaming_for_model_standarized
test <- df_gaming_for_model_standarized
model1_gaming <- spark.logit(training, IsAcceptedAnswer~., maxIter = 100)
#ElasticNetParam = 0 default
```

Tworzymy tabele 3 pokazującą oszacowane wartości współczynników, za pomocą funkcji `summary`.

	Oszacowanie współczynników
(Intercept)	-0.486
AnswerCount	0.037
CommentCount	0.055
Score	-0.094
ViewCount	0.022
avg Reputation Commenting	-0.408
avg Views Commenting	0.127
Reputation Question Owner	-0.660
Views Question Owner	0.060
avg Tag Count	0.181

Tabela 3: Oszacowane wartości współczynników dla modelu logistycznego.

	Dokładność	Czulość	Swoistość	Precyzja	Indeks Youdena
wyniki	0.695	0.907	0.308	0.705	0.214

Tabela 4: Miary opisujące wydajność modelu logistycznego.

Widzimy po tabelce 4, że są osiągane niskie wyniki

Następnie tworzymy model na podstawie drzew decyzyjnych (*decision tree*) używając funkcji `spark.decisionTree`.

```
training <- df_gaming_for_model

# Fit a DecisionTree classification model with spark.decisionTree
model1_dtree <- spark.decisionTree(training, IsAcceptedAnswer ~ .,
```

```
                                    "classification")

# Model summary
summary(model1_dtree)

## Formula:  IsAcceptedAnswer ~ .
## Number of features:  9
## Features:  AnswerCount CommentCount Score ViewCount avg_Reputation_Commenting avg_Vie
## Feature importances:  (9,[0,2,4,5,6,7],[0.006129275292912093,0.006015712310728148,0.0
## Max Depth:  5
##  DecisionTreeClassificationModel: uid=dtc_ba5ec7245ae6, depth=5, numNodes=29, numClas
##   If (feature 6 <= 110.0)
##    If (feature 6 <= 22.0)
##     If (feature 7 <= 311.5)
##      Predict: 1.0
##     Else (feature 7 > 311.5)
##      If (feature 6 <= 2.0)
##       If (feature 2 <= 12.5)
##        Predict: 0.0
##       Else (feature 2 > 12.5)
##        Predict: 1.0
##      Else (feature 6 > 2.0)
##       Predict: 1.0
##    Else (feature 6 > 22.0)
##     Predict: 1.0
##   Else (feature 6 > 110.0)
##    If (feature 4 <= 21.375)
##     If (feature 0 <= 1.5)
##      If (feature 5 <= 164.5)
##       Predict: 1.0
##      Else (feature 5 > 164.5)
##       If (feature 6 <= 133.5)
##        Predict: 1.0
##       Else (feature 6 > 133.5)
##        Predict: 0.0
##     Else (feature 0 > 1.5)
##      If (feature 6 <= 442.0)
##       If (feature 5 <= 123.16666666666666)
##        Predict: 1.0
##       Else (feature 5 > 123.16666666666666)
##        Predict: 0.0
##      Else (feature 6 > 442.0)
##       Predict: 0.0
##    Else (feature 4 > 21.375)
##     If (feature 6 <= 559.0)
##      If (feature 6 <= 161.5)
##       If (feature 2 <= 5.5)
##        Predict: 0.0
```

```
##       Else (feature 2 > 5.5)
##        Predict: 1.0
##      Else (feature 6 > 161.5)
##       Predict: 0.0
##     Else (feature 6 > 559.0)
##      Predict: 0.0
##
```

Istotność zmiennych drzewa decyzyjnego znajduje się poniżej.

	Istotność współczynników
AnswerCount	0.0061290000
CommentCount	0.0000000000
Score	0.0060100000
ViewCount	0.0000000000
avg Reputation Commenting	0.0565290000
avg Views Commenting	0.0011720000
Reputation Question Owner	0.9248900000
Views Question Owner	0.0052700000
avg Tag Count	0.0000000000

Tabela 5: Istotność współczynników dla drzewa decyzyjnego.

	Dokładność	Czulość	Swoistość	Precyzja	Indeks Youdena
wyniki	0.798	0.920	0.576	0.799	0.495

Tabela 6: Miary opisujące wydajność modelu opartego na drzewach decyzyjnych.

Widzimy, że na tabeli 6 widnieją dużo lepsze wyniki, niż na tabeli 4.

Następnie tworzymy model na podstawie lasu losowego (*random forest*) używając funkcji `spark.randomForest`.

```
# Fit a DecisionTree classification model with spark.decisionTree
model1_RF <- spark.randomForest(training, IsAcceptedAnswer ~ .,
                                "classification")

# Model summary
#summary(model1_RF)
```

	Dokładność	Czulość	Swoistość	Precyzja	Indeks Youdena
wyniki	0.794	0.924	0.557	0.792	0.481

Tabela 7: Miary opisujące wydajność modelu opartego na lesie losowym.

Widzimy po tabelce 7, że jest osiągana niższa dokładność niż w tabelce 6.

Z uwagi na to, że najlepszą dokładność uzyskaliśmy dla modelu opartego na drzewach losowych, spróbujemy stworzyć inne modele, na podstawie zmienionych poszczególnych parametrów w funkcji `spark.decisionTree` oraz wyodrębnieniu istotnych zmiennych. Na początku skupimy się na stworzeniu modelu z trzeba najbardziej istotnymi zmiennymi z tabeli 5. Będą to zmienne `Reputation_Question_Owner`. `avg_Reputation_Commenting` i `AnswerCount`.

```
training <- df_gaming_for_model

# Fit a DecisionTree classification model with spark.decisionTree
model1_dtree_significant_variables <- spark.decisionTree(training,
             IsAcceptedAnswer ~
             AnswerCount+avg_Reputation_Commenting
             +Reputation_Question_Owner, "classification")

# Model summary
#summary(model1_dtree_significant_variables)
```

	Dokładność	Czulość	Swoistość	Precyzja	Indeks Youdena
wyniki	0.798	0.920	0.576	0.799	0.495

Tabela 8: Miary opisujące wydajność modelu opartego na drzewach decyzyjnych dla konkretnych zmiennych.

Na koniec skupimy się na stworzeniu modelu ze zmienionym parametrem `maxDepth` $= 3, 7, 9$ (parametr `maxDepth` $= 5$ jest ustawiony domyślnie).

Zaczynamy od `maxDepth`$= 3$.

```
training <- df_gaming_for_model

# Fit a DecisionTree classification model with spark.decisionTree
model1_dtree_significant_maxDepth3 <- spark.decisionTree(training,
                                                         IsAcceptedAnswer ~ .,
                                        "classification", maxDepth = 3)

# Model summary
#summary(model1_dtree_significant_maxDepth3)
```

	Dokładność	Czulość	Swoistość	Precyzja	Indeks Youdena
wyniki	0.796	0.921	0.567	0.795	0.488

Tabela 9: Miary opisujące wydajność modelu opartego na drzewach decyzyjnych dla `maxDepth`$=$ 3.

Następnie `maxDepth`$=$ 7.

```
training <- df_gaming_for_model

# Fit a DecisionTree classification model with spark.decisionTree
model1_dtree_significant_maxDepth7 <- spark.decisionTree(training,
                                                         IsAcceptedAnswer ~ .,
                                        "classification", maxDepth = 7)

# Model summary
#summary(model1_dtree_significant_maxDepth7)
```

	Dokładność	Czulość	Swoistość	Precyzja	Indeks Youdena
wyniki	0.800	0.941	0.544	0.790	0.485

Tabela 10: Miary opisujące wydajność modelu opartego na drzewach decyzyjnych dla `maxDepth`= 7.

Na koniec `maxDepth`= 9

```
training <- df_gaming_for_model

# Fit a DecisionTree classification model with spark.decisionTree
model1_dtree_significant_maxDepth9 <- spark.decisionTree(training,
                                                         IsAcceptedAnswer ~ .,
                                        "classification", maxDepth = 9)

# Model summary
#summary(model1_dtree_significant_maxDepth9)
```

	Dokładność	Czulość	Swoistość	Precyzja	Indeks Youdena
wyniki	0.806	0.941	0.559	0.796	0.500

Tabela 11: Miary opisujące wydajność modelu opartego na drzewach decyzyjnych dla `maxDepth`= 9.

Po tabelkach 11, 10 i 9, że im wyższy parametr `maxDepth` tym większa dokładność jest osiągana.

7 Podsumowanie

Podsumowując wyniki osiągane dla wszystkich rozważanych modeli można dojść do wniosku, że najlepsze wyniki są osiągane dla modeli wykonanych przy pomocy drzew decyzyjnych. W każdym przypadku można zauważyć, że modele osiągają lepsze wyniki przy poprawnym sklasyfikowaniu pozytywnych przypadków (czułość), niż przy poprawnym klasyfikowaniu negatywnych przypadków (swoistość). Przez to wartości Indeksu Yowdena nie były za wysokie.