Sadržaj

Implementacija modela

Uvod

Proces implementacije zahteva rad u R-u i SQL-u. Pretpostavka je da postoji dugme SUBMIT na koje se klikne kada želimo da se izračuna verovatnoća default-a. Kada se klikne na dugme, uneti podaci vezani za aplkaciju se čuvaju u bazi, zatim se poziva EVENT koji bi pozivao R funkciju-u koja vraća izračunatu verovatnoću koja može da se upiše u bazu. Argumenti koji bi se prosleđivali su: id (app), app_version i id (model).

Propratni dokument je kod sa komentarima, odnosno R fajl- DataEngineer_BSANDO.

Uvođenje novih tabela

Kako je moguće da kreditna aplikacija ima više faza, u smislu da klijent ne mora odmah da popuni sve podatke, a želimo da imamo mogućnost da za svaku fazu izračunamo verovatnoću, uvedena je tabela *AppHistory* sa sledećim karaketristikama.

```
CREATE TABLE AppHistory
(
APP_ID integer,
APP_VERSION integer not null, ,--koliko puta se računala verovatnoća jer se klijent
vraćao
SEX  text not null,
FAMILY text not null,
CHILDREN integer  NULL,
RENT text not null ,
REDACTIONDATE date not null,
DEFAULT_PROB  FLOAT NULL,--izračunata verovanoća bi se upisivala ovde
PRIMARY KEY (APP_ID, APP_VERSION),
FOREIGN KEY (APP_ID) REFERENCES socio_demographic(id)
);
```

Tada će svaki klik na dugme SUBMIT (ako su podaci izmenjeni) data novi red u bazi i povećati APP_VERSION za 1.

Primer tabele:

AppHistory

APP_ID	APP_VERSION	SEX	FAMILY	CHILDREN	RENT	REDACTIONDATE	DEFAULT_PROB
123	1	Female	Married	NULL	Y	1.1.2020	0,002
123	2	Female	Married	5	Y	1.10.2020	0,003
124	1	Female	Married	1	N	1.1.2020	0,0001
125	1	Male	Single	NULL	N	1.1.2020	0,0002
125	2	Male	Single	1	N	1.11.2020	0,0001

Vidimo da kreditne aplikacije 123 i 125 imaju dve verzije i samim tim dve različite verovatnoće.

U tabeli socio_demographic uvedene su još dve kolone:

1. APP_VERSION - ima vrednost poslednje verzije kreditne aplikacije, za kredinu aplikaciju 123 izgledala bi kao na slici dole.
2. MODEL_ID – (foreign key ka tabeli MODELS) ima vrednost modela na osnovu kojeg se radi obračun verovatnoće

socio_demographic

ID	SEX	FAMILY	CHILDREN	RENT	APP_VERSION	MODEL_ID
123	Female	Married	5	Y	2	1

Kako bi u bazi imali informaciju o modelima sa kojima banka raspolaže uvodimo tabelu MODELS. Tabela ima sledeće karakteristike.

```
CREATE TABLE MODELS
(
ID integer not null,

NAME varchar(100) not null,

MODEL_FAMILY varchar(100) not null,--ukoliko želimo u budućnosti da vidimo koliko smo modela pravili za verovanoću default-a

MODEL_TYPE varchar(100) not null, --ukoliko želimo u budućnosti da vidimo koliko puta smo koristili određen tip modela

OBSOLETE BIT not null
PRIMARY KEY (ID)
)
```

Primer tabele:

MODELS

ID	NAME	MODEL_FAMILY	MODEL_TYPE	OBSOLETE
1	Defaul Probabilty model	DEFAULT_PROB	LOGIT_REG	FALSE

Kako bi u bazi imali podatke o kolonama koje konfigurišu u modelu uvedena je tabela MODEL_DETAILS. Ideja ja da, kada data scientist obavesti data engineer-a koje kolone konfigurišu u modelu i kakva transformacija treba da se primeni nad njima, tada data engineer može da unese

podatke u ovu tabelu i da definiše način kojim vrši transformacija koju zahteva data scientist i način da se eliminišu loši podaci.

Tabela ima sledeće karakteristike.

```
CREATE TABLE MODEL_DETAILS
(
  model_id integer not null,

  model_version integer not null, --može biti više verzija istog modela

  coef float not null, --koeficijenti koji učestvuju u modelu

  col_name varchar(100) not null, --ime kolone koja učestvuje u modelu, npr OVERDUE
(ime iz baze)

  col_classification varchar(100) not null, /*- ime nove kolone, tj. nove klase koja
  je potrebna data scientist-u.

   Npr, OVERDUE se raščlanjuje na 3 klase: OVERDUE1 (kašnjenje manje od 1
   dan), OVERDUE25...
   FAMILIY postaje marriedTrue i slično.
   Ako se ime kolone koja učestvuje u modelu ne menja u odnosu na ime u bazi, tada
   se uzima vrednost col_name, primer je kolona debt.
   Debt se isto zove i u modelu i bazi, nema potrebe za transformacijom.
*/

   col_type varchar(100) not null,/* -ime tipa kolone (RANGE, MATCH, +,-...)
   Ako se kolona raščlanjava na klase prema:
   - opsegu vrednosti   -> RANGE,
   - stringu            -> MATCH
   ili
   ne raščlanjava se, već vrednost mora biti pozitivna ili negativna, tada vrednost
može biti + ili -

   */
  string_match varchar(max) not null, /*-sa čim kolona mora biti jednaka da bi
vrednost bila 1.
  Npr, marriedTrue će biti
  1 ako je FAMILY = MARRIED, stoga će u ovom polju biti upisana vrednost MARRIED*/

  start_range float not null, --ukoliko se kolona raščlanjava po opsegu, ovde će
pisati donja granica
  end_range float not null,   --ukoliko se kolona raščlanjava po opsegu, ovde će
pisati gornja granica

  PRIMARY KEY (model_id, model_version,col_classification),
  FOREIGN KEY (model_id) REFERENCES MODELS(id)
)
```

Za model korišćen u dev.pdf dokumentu, tabela bi izgleda kao na slici ispod.

MODEL_DETAILS								
MODEL_ID	MODEL_ VERSION	COEF	COL_NAME	COL_CLASSI FICATION	COL_TYPE	STRING_ MATCH	START_ RANGE	END_ RANG
1	1	-5.1626802	COEF	intercept			0	0
1	1	0.0007927	debt	debt	+		0	0
1	1	0.3996028	sex	sexmale	match	Male	0	0
1	1	0.8707857	family	marriedTRUE	match	MARRIED	0	0
1	1	0.3482291	children	children	+		0	0
1	1	1.4557413	rent	rentTRUE	match	Y	0	0
1	1	0.4927653	overdue	overdue_0	range		0	1
1	1	0.9504462	overdue	overdue_25	range		2	25
1	1	1.0687459	overdue	overdue_inf	range		26	10^10

Kontrola kvaliteta podataka

Ideja je da data engineer pri unosu svake kolone koja učestvuje u modelu, unese podatke na osnovu kojih se ta kolona transformiše, a samim tim se i pogrešne vrednosti transformišu.
To se postiže kolonama COL_TYPE, STRING_MATCH, START_RANGE i END_RANGE.

Kolona koja odgovara zahtevu data scientist-a je ***final_col.*** U nju upisujemo vrednosti zahtevane u zadaci.pdf – sekcija 4.2.1.

Ideja je opisana narednim postupkom.

- Data enigneer uneste podatke u MODEL i MODEL_DETAILS.
- U R-u učitamo tabelu MODEL_DETAILS i uzimamo sve jedinstvene vrednosti kolone COL_NAME (osim COEF) i pravimo petlju na osnovu tih vrednosti.

- Kada se podaci se unesu u kreditnu aplikaciju i u tabelu AppHistory, mi ih učitamo u R.

- Kroz petlju koja sadrži imena kolona se krećemo kolonu po kolonu u učitanoj tabeli AppHistory (ima 1 red) i svaku vrednost poredimo sa zadatom vrednošću iz MODEL_DETAILS.

  - U ovom primeru imaćemo (debt, sex, family, children, rent, overdue) u for petlji.
  Prva iteracija uzima vrednost debt iz učitanog reda za kreditnu aplikaciju, npr -8000.
  Pošto je col_type = "+", pitamo da li je vrednost debt NA ili NULL, ako nije u novu kolonu - ***final_col*** napisi vrednost ABS(debt), inace napisi 0.

  - Sledeca iteracija uzima kolonu sex. Posto je col_type='match', pitamo da li je stvarna vrednost kolone sex u listi definisanoj u koloni STRING_MATCH, ako jeste u novu kolonu - ***final_col*** upiši 1, inace 0.
    ...
  - Poslednja iteracija uzma vrednost iz kolone OVERDUE. Ova kolona, po zahtevu data scientist-a mora biti raščlanjena na 3 klase.
    Svaka klasa ima svoj *RANGE.* Pitamo da li stvarna vrednost iz aplikacije pripada nekom range-u, ako da, upisi 1, inace 0.

Izračunavanje verovanoće default-a

Cilj je da nakon for petlje opisane u malopređašnjem postupku dobijemo tabelu sledećeg oblika:

A tibble: 8 x 11

model_version	model_code	coef	col_name	col_class	col_type	string_match	start_range	end_range	final_col
<dbl>	<chr>	<dbl>	<chr>	<chr>	<chr>	<chr>	<dbl>	<dbl>	<dbl>
1	PROF_DEFAULT	0.000793	debt	""	+	""	0	0	8000
1	PROF_DEFAULT	0.400	sex	"sexmale"	match	"Male"	0	0	0
1	PROF_DEFAULT	0.871	family	"marriedTRUE"	match	"MARRIED"	0	0	1
1	PROF_DEFAULT	0.348	children	""	+	""	0	0	4
1	PROF_DEFAULT	1.46	rent	"rentTRUE"	match	"Y"	0	0	0
1	PROF_DEFAULT	0.493	overdue	"overdue_0"	range	""	0	1	1
1	PROF_DEFAULT	0.950	overdue	"overdue_25"	range	""	2	25	0
1	PROF_DEFAULT	1.07	overdue	"overdue_inf"	range	""	26	10000	0

Kolona ***final_col*** ima oblik koji je zahtevao data scientist, odnosno 1 i 0 za kategoričke promenljive i stvarna vrednost ili 0, za numeričke promenljive.

Sada možemo da pomnožimo kolonu ***coef*** sa kolonom ***final_col*** i da izračunamo zbir tih vrednosti. Kako bi dobili *score*, dovoljno je da na taj zbir dodamo početni koeficijent, odnosno kolonu COEF, gde je COL_NAME=COEF (iz tabele MODEL_DETAILS).
Sada, kada imamo skror, možemo lako da ga ubacimo u zadatu formulu i izračunamo verovanoću.

Kada se pozove funkcija ***get_p<-function(model_id, app_id,app_version)*** u R-u, ona prolazi kroz navedene korake i vraća izračunatu verovanoću koja može da se upiše u bazu u tabelu AppHistory, pomoću INSERT komande u SQL-U. Kako znamo id aplikacije i verziju aplikacije, imamo jednistveni ključ potreban za tu tabelu.

Deo koda u R-u:

```
+   #jedinstvene vrednosti kolona koje ucestvuju u modelu
+   distinct_col_name<-unique(filter(current_model, col_name!="COEF")$col_name)
+
+   #prazna tabela u koju cemo upisivati rezultat for petlje
+   model_calc <- filter(current_model, model_version == "-")
+
+   j<-1
+
+   for (i in distinct_col_name) {
+
+     #uzmi redove gde je col_name = ime kolone iz petlje
+     df<-filter(current_model, col_name == i)
+
+     #dodaj novu kolonu koja ima oblik koji zahteva data scientist
+     df<-mutate(df, final_col = 0)
+
+     #prvobitna ideja je bila da vrednostima iz current_app pristupim pomocu pokazivaca - current_app@i (gde je i ime
kolone), medjutim R
+     #ne podrzava pokazivace, pa sam morala da koristim redni broj kolone - current_app[j]
+
+     x<-current_app[j] #vrednost iz aplikacije za kolonu iz petlje
+
+
+     #tip podatka definisan u model_details: range, match, +,-,...
+     curr_col_type<-unique(df$col_type)
```

```
+
+    if(curr_col_type=='range'){
+      x<-as.double(x)
+      df<-mutate(df, final_col=as.double(ifelse(start_range <= x & x<=end_range,  1,0)))
+
+    } else if (curr_col_type=='match') {
+      x<-as.character(x)
+      df<-mutate(df, final_col=as.double(ifelse(length(grep(x,string_match))==1,  1,0)))
+
+    } else if (curr_col_type=='+') {
+      x<-as.double(x)
+      df<-mutate(df, final_col=as.double(ifelse( !is.na(x) & !is.null(x),  abs(x),0)))
+
+    } else if (curr_col_type=='-') {
+      x<-as.double(x)
+      df<-mutate(df, final_col=as.double(ifelse( !is.na(x) & !is.null(x),  -abs(x),0)))
+
+    }
+    model_calc<-bind_rows(model_calc,df)
+    j<-j+1
+
+  }
+
+  #probability
+  a<-(filter(current_model, col_name == 'COEF'))$coef #uzmima se intercept
+
+  #mnoze se koeficijenti sa transformisanom kolonom (u skladi sa data science potrebama)
+  model_calc$coef_col <- model_calc$coef*model_calc$final_col
+
+  score<-a+sum(model_calc$coef_col)
+  p<-1/(1+exp(-score))
+
+  # VRACA SE VEROVATNOCA DEFAULT-A
+  ############
+  return(p) #
+  ############
+ }
```

Zaključak

Na ovakav način smo obezbedili da se za jedan model čuvaju svi koeficijenti, kolone i njihov modifikovan oblik koji zahteva data scientist (tabela MODEL_DETAILS).
Ukoliko dođe do izmene koeficijenta u modelu ili izmene kolona nije potrebno menjati kod, već je dovoljno da data engineer unese izmene u tabelu MODEL_DETAILS i izmeni verziju modela, ukoliko je potrebno. Eventualno da doda jos neki if statement ukoliko se kolona transformiše na specifičan način.
Takođe, data engineer ima kontrolu nad kvalitetom podataka tako što za svaku kolonu definiše kom tipu može pripadati i na osnovu kojih vrednosti se vrši transformacija.