Rating Default Projekat

# Biznis problem

Problem koji treba da rešimo je procena rizika izdavanja bankarskih kredita klijentima. Problem je tipa binarne klasifikacije nebalansiranog skupa podataka, što znači da od 2 moguće klase ciljne promenljive (*target variable*), jedna dominira nad podacima u skupu podataka.

Cilj projekta je da se nadmaše rezultati logističke regresije koju banke trenutno koriste.

Skup podataka nad kojima je rađena analiza i izrada modela je preuzet sa Kaggle-a:

* <https://www.kaggle.com/uciml/default-of-credit-card-clients-dataset>

# Rešenje problema

Za probleme nebalansiranog skupa podataka najčešće se koristi F-score kao mera za evaluaciju modela, I time smo se mi vodili.

Prvobitno je rađena eksplorativna analiza podataka, radi upoznavanja I vršenja analize nad skupom podataka, ispracena raznim vizuelizacijama.

Sledeći korak je podela podataka na skupove podataka za treniranje I testiranje modela (80:20). Čest pristup kod problema nebalansiranog skupa podataka je da se vrši *Sampling* skupa podataka. U ovom problemu je to rađeno na 3 načina:

* *Undersampling* – smanjenje broja podataka dominantne klase, odbacivanjem nasumičnih podataka, kako bi se izbalansirao skup podataka.
* *Oversampling* – povećanje broja podataka nedominantne klase kopiranjem već postojećih nasumičnih podataka, kako bi se izbalansirao skup podataka
* *SMOTE* – sintetičko kreiranje novih podataka kako bi se izbalansirao skup podataka.

Nakon toga vrši se treniranje više različitih modela radi pronalaženja optimalnog. U procesu treniranja modela, vrši se optimizacija hiperparametara i nakon toga je kreiran detaljan *dashboard* u kome se vrši komparacija svih modela prema različitih tehnikama evaluacije modela, prikazani *gain* I *lift* grafici, I odrađen *Expected value framework*.

Na slici 1 se vide rezultati sledećih modela:

* LightGBM – sa podešenim optimalnim threshold-om (pragom klasifikacije)
* Logistička regresija sa izvršenim SMOTE samplovanjem
* Logistička regresija bez ikakve obrade podataka

Graphical user interface, application

Description automatically generated

Slika 1- Komparacija modela

Optimalni *threshold* predstavlja prag klasifikacije koji maksimizira F-score metriku.

Na osnovu prethodne slike zaključili smo da je LightGBM postigao najbolje rezultate, I selektovan je kao optimalni model. Više o njemu može se saznati sa sledećih linkova:

* LightGBM docs:   
  <https://lightgbm.readthedocs.io/en/latest/index.html>
* Treesnip (biblioteka za integraciju lightgbm/catboost u tidymodels):  
  <https://github.com/curso-r/treesnip>
* Detaljan opis parametara:  
  <https://sites.google.com/view/lauraepp/parameters>
* Implementacija LightGBM-a sa Tidymodels uz pomoć Treesnip-a:  
  <https://www.r-bloggers.com/how-to-use-lightgbm-with-tidymodels/>

Nakon selekcije modela, kreiran je API za jednostavan pristup prethodno istreniranom modelu. Za kreiranje API-ja razmatrana su 2 paketa: *plumber* I *openCPU*.

### Plumber

|  |  |
| --- | --- |
| **Prednosti** | **Mane** |
| * Lagan za korišćenje * Može da se koristi kao ekskluzivni back-end interaktivne web app * Plumber filteri koji mogu da se koriste za upravljanje request-ovima * Podržava dinamične rute i tipizirane dinamične rute * Obuhvata statični file server za hostovanje statičnih fajlova (JavaScript, CSS, HTML) * Podržava upravljanje state-om | * Jednonitni (Singlethread) |

### OpenCPU

|  |  |
| --- | --- |
| **Prednosti** | **Mane** |
| * Višenitni (Multithread) * Jednostavna implementacija servera (Linux, rApache) * Širok domen output formata * Isproban i korišćen u više aplikacija * Jednostavan deploy paketa i aplikacija na ocpu cloud serveru * Podržava upravljanje state-om | * Zahteva poznavanje JavaScript-a * Zahteva izradu R paketa * Podržava samo GET i POST metode * Upravljanje state-om na sopstveni način |

*Plumber* se pokazao kao bolji izbor usled jednostavne implementacije, jake podrške i redovnog ažuriranja.

API je kreiran pomoću *plumber* paketa u programskom jeziku *R*-u. Pomoću *docker*-a napravljeno je virtuelno okruženje u kome se pokreće prethodno napravljeni API povezan sa *PostgreSQL* bazom koja se nalazi na server mašini.

## Integracija u aplikaciju

Klijent u *Rating* aplikaciji klikne na odgovarajuće dugme, šalje se *GET Request* na prethodno kreirani API koji započinje predikciju svih klijenata u bazi podataka, za koje nije već utvrđena klasa *default* stanja, I ažurira odgovarajuće kolone.

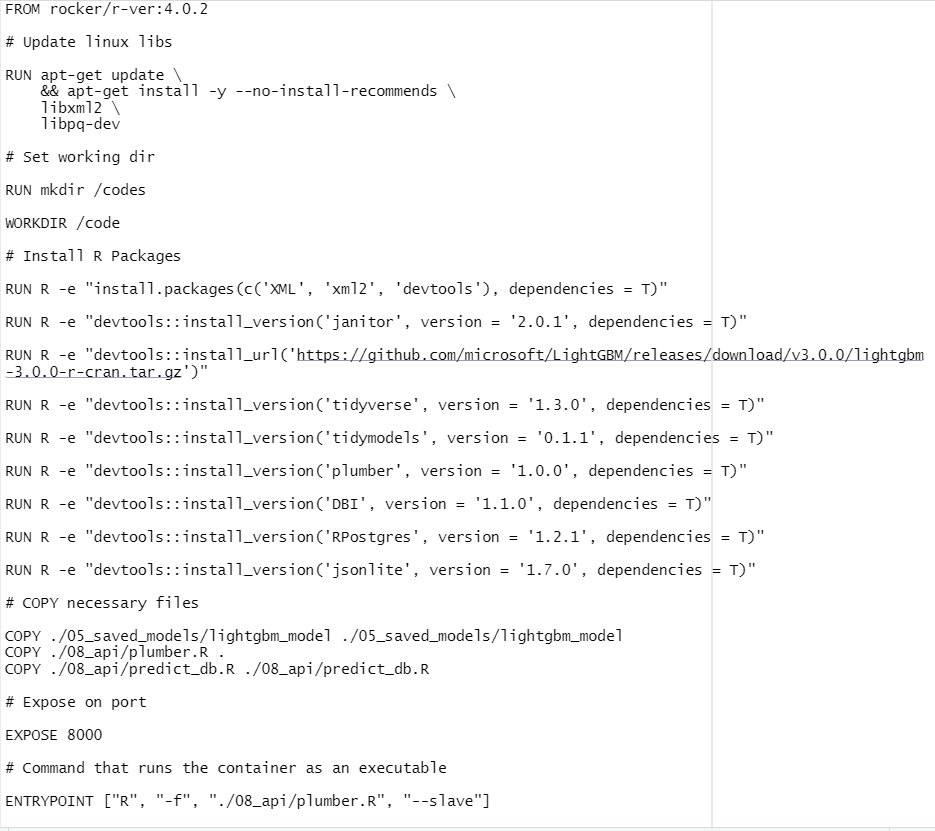
# Workflow

Dockerfile → Building Image → Docker-composer

## Dockerfile

* Kao početnu tačku učitati neki osnovni Docker Image.
* Ažurirati neophodne zavisnosti na linux OS-u
* Instalirati neophodne pakete sa određenim verzijama i njihove zavisnosti
* Kopirati fajlove neophodne za rad API-ja u odgovarajuće direktorijume
* Predložiti port 8000
* Pozvati funkciju koja izvršava plumber.R skriptu u --slave režimu.

(--slave → run R as quietly as possible)



## Kreiranje docker image-a

U direktorijumu u kome se nalazi Dockerfile, pokrenuti terminal i uneti sledeću komandu:

* docker build -t registry.example.com/group/project/image -f DOCKERFILE .

Sačekati da se image napravi, zatim push-ovati image u Container Registry datog projekta

* docker push registry.example.com/group/project/image

## Docker composer

Napraviti docker-composer.yml fajl



Pokrenuti docker composer unošenjem komande u terminal:

* *docker-composer up*

Neophodan nginx usled predugog čekanja response-a sa API-ja, radi modifikacije maksimalnog dozvoljenog vremena za response.

**\*\*NAPOMENA**: Komanda „volumes“ *mount-*uje putanju na trenutnu, i izbacuje se iz upotrebe ukoliko se docker-compose koristi u produkciji.

