

---

# Домашна задача 3

---

Наталија Насева  
221161  
11 јануари 2026 год.

## 1 Offline фаза

### 1.1 Поделба на податочно множество

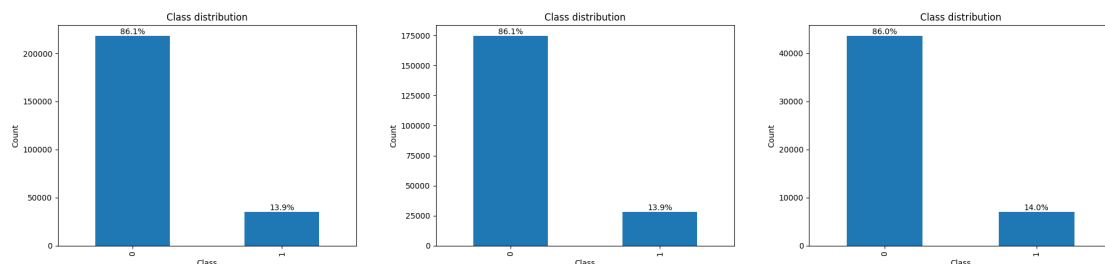
Првиот чекор за припремање на податоците за offline тренирање беше поделба на целото податочно множество на множества за offline тренирање и евалуација и online инференција.

За таа цел, едноставно го делам податочното множество во однос 80% offline и 20% online, со помош на `train_test_split` од `scikit-learn`.

```
train_test_split(df, test_size=0.2, shuffle=True)
```

Бидејќи податоците не содржат временски карактеристики и не се временски подредени, поделбата ја направив со `shuffle=True`.

Важен аспект за поделбата на податоците е одржувањето на иста застапеност на двете класи на променливата која се предвидува (`Diabetes_binary`). Бидејќи во поделбата дозволив податоците да бидат измешани, со пресметување на распределбата воочив дека соодносот на двете класи остана ист во двете нови податочни множества, како што е прикажано на слика 1.1.



(а) Распределба на класите во оригиналното податочно множество (б) Распределба на класите во offline податочното множество (в) Распределба на класите во online податочното множество

Слика 1.1: Распределба на класите на Diabetes\_binary

## 1.2 Избор и тренирање на модел

Следнот чекор во offline фазата е избор на соодветен модел за моделирање на податоците.

Избрав три модели за евалуација, со различно ниво на комплексност:

1. Logistic Regression (`LogisticRegression`)
2. Random Forest Classifier (`RandomForestClassifier`)
3. Gradient-Boosted Trees Classifier (`GBTCClassifier`)

Целта со изборот на модели ми беше да тестирам различни модели кои имаат различно ниво на експресивност и баланс на bias и variance.

Моделот за логистичка регресија главно служеше како baseline со кој што можев да ги споредувам другите два модели.

Random Forest моделот е многу помокен од логистичка регресија и многу поуспешно ги моделира нелинеарните зависности во податочното множество.

Конечно, Gradient-Boosted Trees моделот е најсуфистицираниот од трите избрани модели, кој најдобро ги моделира зависностите во податочното множество, но е најмалку интерпретабилен.

### 1.2.1 Вчитување и претпроцесирање на податоците

Пред да можам да ги евалуирам перформансите на избраните модели, морав да ги вчитам и претпроцесирам податоците за тие да бидат во соодветен формат за моделите.

Бидејќи сите карактеристики во податочното множество, освен BMI, беа категоријски карактеристики (или бинарни или ординални), тие едноставно ги вчитав како `IntegerType` податоци, додека пак BMI како `DoubleType`, со помош на функцијата `cast_features_and_target`.

Понатаму, како дел од pipeline-от за евалуација на моделите воведов и чекор за претпроцесирање.

Овој чекор едноставно креира `VectorAssembler` за моделите базирани на дрва на одлука (`RandomForestClassifier` и `GBTCClassifier`), додека пак за моделот за логистичка регресија дополнително ги скалира податоците. Одлучив да ги скалирам (со `StandardScaler`) податоците кои се на влез на моделот за логистичка регресија бидејќи се оптимизира со помош на методи базирани на пресметување на градиенти, па важно е сите карактеристики да имаат исто влијание. Ова го постигнав со функцијата `get_preprocessing_stages`.

### 1.2.2 Избор на хиперпараметри

За секој од моделите избрав множество од хиперпараметри кои имаат најголемо влијание на квалитетот на предвидување на моделот, а за секој од параметрите избрав вредности кои се соодветни на податоците. Ова го постигнав со `ParamGridBuilder` од Spark и функцијата `build_models`.

За `LogisticRegression`:

- `regParam` (контролира колку силно се казнуваат коефициентите) - со вредности {0.01, 0.1}
- `elasticNetParam` (балансира L1 и L2) - со вредности {0.0, 0.5}

За `RandomForestClassifier`:

- `numTrees` (балансира bias и variance) - со вредности {30, 50, 100}
- `maxDepth` (комплексност на модел / максимална длабочина) - со вредности {5, 8, 10}

За `GBTCClassifier`

- `maxDepth` (комплексност на модел / максимална длабочина) - со вредности {3, 5, 8}
- `stepSize` (стапка на учење) - со вредности {0.05, 0.1}

### 1.2.3 Евалуација и избор на најдобар модел

Конечно, за евалуација ја искористив метриката F1 score, со помош на крос-валидација со 5 преклопувања. За ова ја искористив функцијата `train_and_select_best_model`.

Модел	F1
LogisticRegression	0.8278
RandomForest	0.8233
Евалуација	0.8338

Табела 1.1: F1 score за секој од евалуираните модели

Според добиените резултати, прикажани на табела 1.1, сите модели постигнаа слични перформанси (што се должи повеќе на податочното множество отколку на моделите), со тоа што GBT моделот имаше највисок f1 score.

Како последен чекор од offline фазата, го тренирав најдобриот модел на сите податоци од offline податочното множество.

## 2 Online фаза

Online фазата може да се подели на три главни дела:

1. Producer
2. Online предвидување
3. Consumer

### 2.1 Producer

За да симулирам испраќање на податоци на ниво на пациент, од креираното online податочно множество, прво ја отстранувам target колоната, и ги делаам податоците на ниво на запис. Така поделените податоци, во json формат ги испраќам на `diabetes_data` Kafka topic, со пауза од една секунда помеѓу секој запис.

### 2.2 Online предвидување

Online предвидувањето е поделено во три скрипти, `prediction.py`, `kafka_io.py` и `inference.py`.

Главната функција на `prediction.py` е оркестрирање на останатите компоненти кои се потребни за вчитување на податоците, повикување на моделот и враќање на предвидените вредности.

#### 2.2.1 `kafka_io.py`

`kafka_io.py` е одговорна за сите операции кои се поврзани со Kafka streams. Функцијата `read_kafka_stream` иницијализира DataFrame кој е поврзан со Kafka topic-от на кој producer-от ги испраќа влезните податоци. Избирам пораките да се консумираат почнувајќи од најраниот достапен офсет со цел поедноставно тестирање.

`parse_json` е одговорна за вчитување на податоци во соодветен формат, од влезниот key-value бинарен формат, во шема на податоци на која беше трениран моделот, со помош на Spark StructType `DIABETES_SCHEMA`.

Конечно, функцијата `write_stream` повторно ги враќа податоците во json формат, но овој пат заедно со предвидената класа и ги испраќа на Kafka topic-от кој го чита consumer скриптата.

### 2.2.2 inference.py

Задачата на оваа скрипта е да го вчита зачуваниот модел кој беше трениран на offline податоците (`load_model`), со него да изврши предвидување на вчитаните податоци од Kafka stream-от, и повторно да ги врати податоците кои му беа испратени на моделот, заедно со предвидената класа `run_inference`.

Излезните податоци се во следниот формат:

```
{
  "HighBP": 0,
  "HighChol": 0,
  "CholCheck": 1,
  "BMI": 24.0,
  "Smoker": 0,
  "Stroke": 0,
  "HeartDiseaseorAttack": 0,
  "PhysActivity": 1,
  "Fruits": 1,
  "Veggies": 1,
  "HvyAlcoholConsump": 0,
  "AnyHealthcare": 1,
  "NoDocbcCost": 0,
  "GenHlth": 1,
  "MentHlth": 1,
  "PhysHlth": 0,
  "DiffWalk": 0,
  "Sex": 1,
  "Age": 7,
  "Education": 6,
  "Income": 8,
  "predicted_diabetes": 0
}
```

## 2.3 Consumer

Скриптата за consumer-от на податоци од `diabetes_predictions` Kafka topic едноставно ги вчитува и прикажува испратените податоци, вклучувајќи ги и новите предвидувања.

## 3 Дополнителни компоненти

### 3.1 Docker Image за Spark

Како дел од домашната креирава и Dockerfile кој дефинира околина во која може да се извршува предвидувањето со моделот, со цел да обезбедам конзистентност на

зависности и поедноставно тестирање.