Домашна 3 РНМП 221126

Изработила: **Ана Андовска 221126**

1. Вовед

Целта на оваа домашна задача е имплементација на комплетен систем за машинско учење кој комбинира **offline фаза за обучување на модели** и **online фаза за обработка и предвидување на податоци во реално време**, користејќи **Apache Spark** и **Apache Kafka**.

Во рамки на задачата се користи **Diabetes Health Indicators Dataset (BRFSS 2015)**, при што со податоците од offline фазата се обучуваат повеќе модели за бинарна класификација (дијабетес / нема дијабетес), а најдобриот модел се користи за предвидување врз поток од податоци во online фазата.

2. Поделба на податоците (offline / online)

Оригиналното податочно множество е поделено случајно на две подмножества:

- **offline.csv** – 80% од податоците (202,944 записи)
- **online.csv** – 20% од податоците (50,736 записи)

Поделбата е извршена со **задржување на соодносот на класите (stratified split)**, со цел да се избегне нарушување на распределбата на класите.

2.1 split_dataset.py – подготовка и поделба на податоците

Првата Python скрипта во pipeline-от е `split_dataset.py`, чија задача е да го подели оригиналното податочно множество на offline и online делови. Оваа поделба претставува основа за раздвојување на фазата на обучување од фазата на обработка во реално време.

```
X_train, X_test = train_test_split(
    data,
    test_size=0.2,
    stratify=labels,
    random_state=42
)
```

Овој чекор завршува без грешки, што укажува на правилна поделба на податоците.

Овој дел од кодот користи `train_test_split` со вклучена стратификација, со што се обезбедува дека соодносот помеѓу класите „DIABETES“ и „NO_DIABETES“ останува ист и во двете датотеки. Ова е особено важно поради класната нерамнотежа во податоците. Фиксниот `random_state` е избран со цел репродуктивност на резултатите.

Резултатот од оваа скрипта е потврден во конзолниот излез, каде што јасно се гледа дека се генерирани 202,944 записи за offline фазата и 50,736 записи за online фазата
Овој чекор завршува без грешки, што укажува на правилна поделба на податоците.

```
> split_dataset.py
Wrote 202944 rows to data\offline.csv
Wrote 50736 rows to data\online.csv
```

3. Offline фаза – Spark имплементација и претпроцесирање

Во offline фазата, датотеката `offline.csv` се вчитува во **Apache Spark апликација** како DataFrame.

Врз податоците се извршуваат следните трансформации:

- селекција на релевантни колони

- стандардизација на нумерички атрибути (StandardScaler)
- креирање на feature vector погоден за Spark ML алгоритми

Сите трансформации се имплементирани како **методи што можат повторно да се применат**, што е од клучно значење за online фазата, каде што мора да се користи идентична логика за обработка.

3.1. offline_train.py – offline обучување и избор на модел

Датотеката `offline_train.py` ја реализира целокупната offline фаза на системот. Нејзината примарна задача е да ги обработи податоците, да обучи повеќе модели за класификација и да го избере најдобриот модел според соодветни евалуациски метрики.

Во почетниот дел од скриптата се вчитуваат податоците од датотеката `offline.csv`. Податоците се разделуваат на влезни карактеристики и целна променлива (`Diabetes_binary`), што овозможува јасна дефиниција на проблемот како задача за бинарна класификација. Следно, податоците се делат на тренинг и тест (80% и 20%) подмножества со задржување на класниот сооднос, со што се обезбедува репродуктивна и фер евалуација.

3.2 Претпроцесирање и скалирање

```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_train)
```

Стандардизацијата е применета бидејќи различните здравствени индикатори имаат различни размери. Овој чекор е особено важен за Logistic Regression, но е задржан и за другите модели со цел конзистентност на pipeline-от.

4. Модели за класификација и параметри

Во offline фазата се обучени три различни модели за бинарна класификација:

1. **Logistic Regression**
2. **Random Forest Classifier**
3. **Gradient Boosted Trees (GBT)**

Logistic Regression

```
lr = LogisticRegression( featuresCol="features", labelCol="label" )
```

Random Forest

```
rf = RandomForestClassifier( numTrees=100, maxDepth=10, featuresCol="features",
 labelCol="label" )
```

Gradient Boosted Trees

```
gbt = GBTClassifier( maxIter=50, maxDepth=5, featuresCol="features", labelCol="label" )
```

За секој модел се тестирани **различни комбинации на хиперпараметри**.

Како метрика за евалуација се користи **F1-score**, при што се анализираат:

- weighted F1-score (поради класна нерамнотежа)
- F1-score за позитивната класа (DIABETES)

За избор на најдобар модел е користена **cross-validation стратегија**.

5. Резултати од offline експериментите

Резултатите од cross-validation се следни:

Модел	Weighted F1	Positive F1	Precision	Recall
Logistic Regression	0.7712	0.4441	0.3127	0.7660
Random Forest	**0.8415**	0.4204	0.4347	0.4070
Gradient Boosted Trees	0.8350	0.2662	0.5812	0.1726

Точно што добив:

```
(venv)
> python src/offline_train.py
[CV] Model=logistic_regression weighted-F1=0.7712 pos-F1=0.4441 P=0.3127 R=0.7660
[CV] Model=random_forest      weighted-F1=0.8415 pos-F1=0.4204 P=0.4347 R=0.4070
[CV] Model=gbt                weighted-F1=0.8350 pos-F1=0.2662 P=0.5812 R=0.1726
================ RESULTS ================
Selected best model (by weighted-F1): random_forest
Test weighted-F1 (thr=0.50): 0.8407 | pos-F1: 0.4223 P: 0.4302 R: 0.4147
Test weighted-F1 (thr=0.393): 0.8199 | pos-F1: 0.4537 P: 0.3686 R: 0.5897
Confusion matrix (thr=0.393): TP=3335 FP=5712 FN=2320 TN=29222
=========================================
Best model saved to models\best_model.pkl
Scaler saved to models\scaler.pkl
Feature names saved to models\feature_names.txt
Info saved to models\best_model_info.json
```

Избор на најдобар модел

Како најдобар модел е избран **Random Forest**, бидејќи покажува највисока вредност на weighted F1-score.

Дополнително е извршена оптимизација на decision threshold, при што вредноста **0.393** овозможува подобар recall за позитивната класа.

---

6. Финална евалуација и confusion матрица

Евалуациски метрики при threshold = 0.393

Метрика	Вредност
Weighted F1-score	0.8199
F1-score (DIABETES)	0.4537
Precision	0.3686
Recall	0.5897

Confusion матрица при threshold = 0.393

	Предвидено: NO_DIABETES	Предвидено: DIABETES
Вистинско: NO_DIABETES	TN = 29222	FP = 5712
Вистинско: DIABETES	FN = 2320	TP = 3335

Овие резултати покажуваат дека моделот успешно идентификува значителен дел од пациентите со дијабетес.

7. Серијализација на моделот

Најдобриот модел и потребните артефакти се зачувани локално
во директориумот `models/`:

- обучен модел (`best_model.pkl`)
- scaler (`scaler.pkl`)
- листа на features (`feature_names.txt`)

- информации за моделот (`best_model_info.json`)

Ова овозможува повторно користење на моделот без повторно тренирање.

8. utils.py – Заеднички помошни функции

Датотеката `utils.py` содржи помошни функции кои се користат и во offline и во online фазата. Нејзината улога е да ја намали дупликацијата на код и да обезбеди конзистентна примена на истите операции во различни делови од системот.

Функцијата `load_pickle` служи за вчитување на серијализирани Python објекти од диск, како што се обучениот модел и scaler-от. Ова овозможува чист и унифициран начин за работа со pickle датотеки без повторување на истата логика во повеќе скрипти.

Функцијата `apply_scaler` ја имплементира логиката за стандардизација на нумеричките карактеристики во online фазата. Наместо повторно да се користи `fit_transform`, што би било некоректно, се користат параметрите `mean_` и `scale_` од веќе обучениот `StandardScaler`. На овој начин се гарантира дека податоците во online фазата се трансформираат идентично како и во offline фазата, што е критично за коректна инференца.

9. Online фаза – Kafka и Spark Structured Streaming

9.1 Kafka producer

Датотеката `producer.py` има задача да симулира реален проток на податоци. Таа ги чита редовите од `offline.csv`, ја отстранува целната променлива и ги испраќа останатите атрибути како JSON пораки во Kafka topic-от `health_data`.

Испраќањето на податоците во JSON формат е избрано поради неговата универзалност и лесна интероперабилност со различни системи. Вметнатиот временски интервал помеѓу пораките овозможува контролирано темпо на стриминг и подобра визуелна демонстрација на работата на системот.

Оваа датотека претставува извор на податоци за online фазата и овозможува реалистична симулација на сценарио во кое податоците пристигнуваат континуирано.

```
producer.send( "health_data", json.dumps(record).encode("utf-8") )
```

Полето што ја означува класата не се испраќа, со што се симулира реален проток на податоци.

10. docker-compose.yml – Kafka и Zookeeper инфраструктура

Датотеката `docker-compose.yml` служи за подигнување на Apache Kafka и Zookeeper со користење на Docker. Ова овозможува брзо и репродуктивно поставување на инфраструктурата без рачна конфигурација.

Со користење на Docker Compose, Kafka се стартува со соодветни порти и listener конфигурација, што овозможува и producer-от и Spark апликацијата да комуницираат со брокерот без дополнителни подесувања.

11. start.sh – Автоматизација на извршувањето

Датотеката `start.sh` претставува shell скрипта за автоматизација на целиот процес. Таа креира Python виртуелна околина, ги инсталира потребните зависности, ја стартува Kafka инфраструктурата и го пушта producer-от.

Оваа скрипта е наменета за олеснување на тестирањето и демонстрацијата, а не како продукциско решение. Нејзината улога е да овозможи брзо стартување на целиот систем со една команда.

12. Spark Structured Streaming consumer

Во online фазата, податоците од датотеката online.csv се испраќаат ред по ред во Kafka topic преку Python producer скрипта. Податоците се испраќаат во JSON формат, при што класата на пациентот намерно не се вклучува, со цел да се симулира реален проток на податоци во кој исходот не е познат однапред.

Во посебна Apache Spark апликација се користи Spark Structured Streaming за вчитување на податоците од Kafka topic-от. Врз секој пристигнат запис се применуваат истите трансформации како во offline фазата, по што записот се предава на претходно обучениот модел за да се добие предвидување. Записите потоа се збогатуваат со предвидената класа и

соодветната веројатност и се испраќаат кон нов Kafka topic, што овозможува понатамошна обработка или анализа од други системи.

12.1 Објаснување на `streaming_app.py` (online фаза)

Датотеката `streaming_app.py` ја имплементира online фазата на системот и служи за предвидување во реално време врз податоци кои пристигнуваат преку Apache Kafka. Апликацијата користи Spark Structured Streaming за да чита податоци од Kafka topic-от `health_data`, да ги обработи на ист начин како во offline фазата и да генерира предвидување за дијабетес за секој пристигнат запис.

Во почетокот, апликацијата ги вчитува резултатите од offline фазата , обучениот модел, scaler-от, листата на feature имиња и оптимизираниот threshold. Ова е важно за да се обезбеди дека online фазата користи **исти трансформации и ист модел** како и при обучувањето.

```
model, scaler, threshold = load_model_scaler_threshold(
    model_path, scaler_path, info_path )
```

Потоа се дефинира Spark сесија и Kafka изворот од кој ќе се читаат податоците. Податоците пристигнуваат како JSON пораки и се парсираат во табеларна форма со соодветна шема.

```
raw_df = spark.readStream.format("kafka") \
    .option("subscribe", input_topic) \
    .load()
```

Следниот чекор е подготовка на влезните карактеристики. Доколку некоја вредност недостасува, таа се заменува со 0.0, со што се спречуваат грешки при инференцата.

```
features_df = parsed_df.select(
    *[F.coalesce(F.col(c), F.lit(0.0)).alias(c)for cin feature_names] )
```

Предвидување (UDF)

Предвидувањето се врши преку user-defined function (UDF). Во неа, податоците се скалираат со истите параметри како во offline фазата, а потоа се користи scikit-learn моделот за да се добие веројатност и класа.

```
prob = model.predict_proba(x_scaled)[0][1]
pred = 1.0 if prob >= threshold else 0.0
```

Резултатот е:

- `prob_diabetes` – веројатност за дијабетес
- `predicted_diabetes` – финална класа (0 или 1)

Овие полиња се додаваат кон секој запис и податоците се испраќаат во нов Kafka topic `health_data_predicted`.
Spark Structured Streaming работи во **микро-бачеви**. Во оваа апликација, за полесна демонстрација, резултатите се печатат на конзола со ограничување од **20 редови по batch**.

Резултатите се испраќаат кон нов Kafka topic: health_data_predicted.

Резултатите се збогатуваат со предвидената класа и веројатноста и се испраќаат кон нов Kafka topic `health_data_predicted`.

Конзолниот излез од Spark апликацијата јасно прикажува дека моделот работи коректно и генерира смислени предвидувања:

```
[Spark] threshold=0.393 | PRINT_CONSOLE=1
-------------------------------------------
Batch: 1
-------------------------------------------
+----+-----------+------+
```

```
|pred|label     |prob  |
+----+----------+------+
|1   |DIABETES  |0.8247|
|0   |NO_DIABETES|0.0186|
|0   |NO_DIABETES|0.1172|
|0   |NO_DIABETES|0.0124|
|0   |NO_DIABETES|0.1629|
|0   |NO_DIABETES|0.0529|
|0   |NO_DIABETES|0.1044|
|0   |NO_DIABETES|0.0154|
|0   |NO_DIABETES|0.1219|
|1   |DIABETES  |0.6282|
|0   |NO_DIABETES|0.0516|
|0   |NO_DIABETES|0.042 |
|0   |NO_DIABETES|0.1515|
|0   |NO_DIABETES|0.0795|
|0   |NO_DIABETES|0.0365|
|0   |NO_DIABETES|0.106 |
|0   |NO_DIABETES|0.0394|
|1   |DIABETES  |0.7661|
|0   |NO_DIABETES|0.0124|
|0   |NO_DIABETES|4.0E-4|
+----+----------+------+
```

---

13. Објаснување на `consumer.py` (визуелизација на резултатите)

Датотеката `consumer.py` не учествува во машинското учење, туку служи **исклучиво за прикажување на резултатите на убав и разбирлив начин**. Таа чита пораки од Kafka topic-от `health_data_predicted`.

```
consumer = KafkaConsumer( "health_data_predicted", bootstrap_servers=["localhost:9092"] )
```

Секоја порака се парсира од JSON и се печати **линија по линија**, на пример:

```
prediction=1 (DIABETES) | prob=0.8247
prediction=0 (NO_DIABETES) | prob=0.0186
```

Ова овозможува:

- јасно да се види како моделот предвидува за секој пациент
- лесна демонстрација на работата на системот
- проверка дали streaming фазата функционира коректно

14. Заклучок

Со оваа домашна задача е успешно имплементиран комплетен **offline–online систем за машинско учење** базиран на Apache Spark и Apache Kafka.

Проектот демонстрира:

- правилна поделба на податоци
- обучување и избор на модел
- серијализација и повторна употреба
- реалновременска обработка на податоци

Архитектурата и имплементацијата се во согласност со современите практики за ML системи во продукција.