Домашна работа 3

Apache Spark & ML

Во оваа домашна задача имплементирав решение налик на MLOps со тренирање на модел и негово сервирање со цел добивање на предвидувања во вистинско време.

Најпрво, го преземав податочното множество Diabetes Indicators кое содржи повеќе нумерички податоци, претходно енкодирани категориски податоци, како и ознака 1/0 за тоа дали пациентот има/нема дијабетес. Податочното множество го исчистив од празни вредности и го поделив на два дела, offline и online, кои ќе се користат соодветно во offline и online фазата од задачата во сооднос 80:20 со задржување на балансот помеѓу позитивната и негативната класа како во оригиналното множество.

Во offline делот, направив уште една дополнителна поделба на offline податочното множество на тренирачко и тестирачко во сооднос 80:20. Исто така, ги одделив колоните со вредностите за карактеристиките од лабелите и на тренирачкото и на тестирачкото подмножество. За скалирање на податоците употребив RobustScaler, кој ова го прави користејќи медијана и интерквартилен опсег (IQR). Се одлучив за овој scaler бидејќи истиот ќе нема проблем при скалирање на невидени вредности кои се надвор од рангот на оние со кои е претходно трениран. Целта на offline фазата е да се истренираат и евалуираат повеќе модели и да се избере најдобриот од нив, кој ќе го сервираме во online фазата. Споредив три класификатори (XGBoost, Random Forest и Logistic Regression) преку GridSearchCV, тестирајќи различни хиперпараметри за секој модел со 5-кратна крос-валидација и оптимизирајќи ја F1 метриката. По пронаоѓањето на најдобрите параметри за секој модел, го споредив нивниот резултат за F1 метриката и го серијализирав најдобриот во .joblib датотека, а тоа беше XGBoost класификаторот. На сличен начин го серијализирав и scaler-от за идна употреба.

Потоа, со помош на start.bat скриптата креирав виртуелна околина, ги инсталирав сите пакети излистани во requirements.txt датотеката, ги кренав потребните docker container-и и ја стартував produce_messages.py скриптата. Тука се читаат податоци од online податочното множество, се форматираат записите (исклучувајќи ја колоната "Diabetes_binary" бидејќи тука се сместени лабелите) и се испраќаат еден по еден како JSON-пораки во Kafka topic наречен "health_data" со пауза од 15 секунди помеѓу пораките.

Во produce_predictions.py скриптата конфигурирав PySpark со Java и Python патеки, стартував SparkSession за обработка на податоци во реално време од Kafka, и ги вчитав претходно истренираните scaler и XGBoost модел. Податоците се читаат од Kafka topic-от "health_data", се обработуваат со Spark Streaming, се трансформираат во соодветен формат и се проследуваат низ UDF-функција која

врши предвидувања со ML моделот. На крај, резултатите (заедно со предвидената класа) се праќаат кон нов Kafka topic "health_data_predicted" со outputMode "append", што значи дека новите предвидувања се додаваат без да ги менуваат претходните податоци, а streaming-от останува активен со query.awaitTermination().

На крај, во consume_messages.py скриптата конфигурирав KafkaConsumer кој слуша пораки од Kafka topic-от "health_data_predicted", ги чита и ги печати на конзола. Consumer-от е конфигуриран да започне од најновите пораки (auto_offset_reset='latest').

Оваа домашна задача покажува како може да се автоматизира целиот процес на машинско учење – од претпроцесирање на податоци и тренирање на моделот до негово интегрирање во систем кој работи во реално време. Со комбинирање на Spark Streaming и XGBoost, овозможив ефикасно обработување на податоци и динамичко генерирање на предвидувања.