VICUNA ZADATAK

Predikcija financijskog instrumenta – Filip Matić

1. Analiza i priprema podataka

Za zadatak predviđanja povrata sam dobio sirove OHLCTV vrijednosti koje su bile relativno pripremljene i čiste. S obzirom da sam odlučio da neću raditi s neuronskim mrežama, bilo je potrebno sirove podatke pretvoriti u konkretne značajke iz kojih algoritmi mogu učiti.
Podaci su bili u odgovarajućim formatima, stoga nije bilo potrebe za pretvorbom.
Kroz sve stupce je bilo mnogo NaN ćelija stoga nisam mogao samo izbaciti sve te podatke, nego sam ih popunio s *interpolate(method="linear").* Tako sam popunio prazne ćelije s linearnom vrijednosti susjednih vrijednosti, s tim da su susjedne vrijednosti jako blizu jedna drugoj. Duplikata nije bilo, tako da sam podatke sortirao po datumu i krenuo na daljnje korake.

1.a. Vizualizacije

Što se tiče vizualizacija, iskreno nisam kontekstualno bio upoznat najbolje s nekim podacima stoga nisam previše eksperimentirao s njima. Nacrtao sam neke osnovne vizualizacije kako bi pronašao korelacije između sirovih podataka i kreirao značajke.
Promatrao sam kretanje cijene kroz vrijeme, trgovački volumen kroz vrijeme, odnos između volumena i povrata, kretanje volatilnosti i distribuciju satnih povrata. Nakon odrađenog osnovnog vizualiziranja, mogao sam početi raditi na značajkama. S obzirom da sam dobio u zadatku ponuđene značajke, a nisam bio najbolje upoznat s financijskom terminologijom, odlučio sam pripremiti skoro sve ponuđene, i još neke dodatne značajke. Lako im je kasnije testirati važnost pa ih filtrirati. Što se tiče izvučenih zaključaka iz vizualizacija, jasno su vidljivi trendovi, poput pada cijene nakon naglog rasta,pa bi pomični prosjeci (EMA i SMA) i momentum trebali biti korisni. Nadalje, iz pregleda trgovačkog volumenta kroz vrijeme vidimo da bi promjena volumena kroz vrijeme trebala pomoći. Od daljnjih vizualizacija kroz bilježnicu sam samo koristio histograme za pregled važnosti značajki.

1.b. Značajke

S obzirom da sam u samom zadatku dobio prijedloge značajki, bilo ih je relativno lako kreirati. Return kroz n dana sam radio sa shift(n), a sma_n s rolling(n).mean, koliko sam razumio što te značajke zapravo znače. Volatility sam napravio s rolling(n).std(), kako bi model bolje razumio oscilacije.
Return značajke spadaju pod „lagged returns", koji modelu prikazuju što se nedavno dogodilo.
Pokretni prosjeci mu tumače dugoročne trendove. Volatility pokazuje koliko cijena oscilira. Omjeri cijena predstavljaju odnos cijena unutar istog sata, a momentum pokazuje koliko se cijena promijenila u odnosu na ranije razdoblje.
Osim navedenih značajki u zadatku,online istraživanjem nisam naišao na neke značajke koje bi još mogle biti bitne u ovom kontekstu.

Naprednije tehničke metrike, poput RSI i MACD, sam naknadno kreirao ali su se pokazale kao nedovoljno korisne pa ih nisam koristio u konačnom modelu. Takve metrike se bolje pokazuju na veće vremenske periode, a ne u ovakvim sat-sat podacima.

Zasad za treniranje modela zadržavam sve navedene značajke, pa ću istražiti koliko su koje bitne za koji model.

2. Modeliranje i evaluacija

Što se tiče modela, zbog vrste podataka i značajki koje su kreirane, odlučio sam se na testiranje sljedećih modela:

Linear Regression
Random Forest Regression
XGBoost
LSTM

Gledajući vizualizacije i podatke, pretpostavio sam da bi najbolje trebali funkcionirati algoritmi koji u obzir uzimaju nelinearnost ovih podataka, odnosno skoro pa nasumičan rast cijene, povrata i volumena prodanog instrumenta.

Testirao sam prvo obični algoritam linearne regresije, iako od njega nisam očekivao nikakve pozitivne rezultate. Linearna regresija nije dovoljan algoritam za upratiti nasumičnost podataka, ali sam ga svejedno uradio da bi vidio napredak kod sljedećih modela. Zadržao sam model linearne regresije jer sam kasnije testirao slagati modele kako bi ih unaprijedio.

Što se tiče podjele na skupove, kod vremenskih slijedova redoslijed podataka ima značenje, pa nisam smio nasumično miješati podatke. Također, mora se učiti iz prošlosti i testirati na budućim podacima. Korištenjem time-based podjele se sprječava curenje podataka.

Sve modele, s obzirom da su regresijski, sam testirao sa sljedećim metrikama:
Mean Absolute Error
Root Mean Squared Error
R-Squared

Nakon linearne regresije, testirao sam „ensemble“ algritme, RandomForestRegressor i XGBoost algoritam.

Za RandomForest, s obzirom da se može eksperimentirati s brojnim metrikama, testirao sam ih koristeći GridSearchCV i RandomizedSearchCV. GridSearchCV bi bio dobra opcija da sam imao računalo s grafičkom karticom, ali sam na raspolaganju imao samo snagu CPU-a. GridSearchCV je bolja opcija za testiranja parametara, ali na većem broju parametara se testira i po stotinjak modela, što je za moje trenutne mogućnosti nerealno. Stoga, sam testirao i RandomizedSearch koji uzima nasumične parametre za testiranje, pa sam testirao zapravo desetak modela. GridSearchCV koji je testirao 30 fitova, ako se dobro sjećam (ne isplati mi ga se ponovno pokrenuti) mi je dao bolje rezultate od metrika koje mi je RandomizedSearch dao. Tako da, za parametre sam koristio
max_depth=10

min_samples_leaf=1
min_samples_split=5
n_estimators=100

RandomForest je dao nešto bolje rezultate, ali vjerujem da bi postigao bolji rezultat kada bi mogao testirati veći broj fitova s GridSearchom.

Još sam testirao XGBRegressor, od kojeg nisam testirao da bude napredniji od RFRegressora ali sam svejedno pokušao ako nisam vidio nešto u podacima. Za XGBoost sam također koristio RandomizedSearchCV, te sam s njegovim parametrima dobio rezultate blago lošije od RandomForesta.

Nakon ensemble modela, htio sam testirati bar jedan sekvencijski model, u ovom slučaju LSTM arhitekturu. Prednost LSTM arhitekture je bila što treba bolje pamtiti sekvence od ostalih modela. Za LSTM je trebalo preoblikovati podatke iz tabličnog formata u 3D oblik. Početni model je bio dovoljno jednostavan da brzo trenira, ali sposoban učiti vremenske obrasce. Testirao sam par puta s različitim vremenskim kontekstom (TIME_STEPS),epohama i „batch“ veličinom te sam naknadno dodao još jedan LSTM sloj. LSTM nije dao dobre rezultate, zato što su podaci od sata do sata praktički pa nasumično varirali, pa LSTM nije mogao pronaći ništa značajno za pamtiti kako bi dovoljno utjecalo na model. Rezultati svih modela su prikazani u sljedećoj tablici

	LinearRegressor	**RFRegressor**	**XGBoost**	**LSTM**
MAE	0.003186	0.003164	0.003165	0.003413
RMSE	0.004878	0.004856	0.004859	0.005135
R2**	-0.009	0.0001	-0.0006	-0.1184

Iako nijedan model nije dao fantastične rezultate, zasad ću se zadovoljiti s RandomForestRegressor algoritmom. Eksperimentirao sam sa stacking metodom kombinirajući linearni i ensemble model. Stacking model je dao gore rezultate od RandomForesta tako da sam zadržao njega za rad na API-u.

Da konkretno odgovorim na pitanja iz zadatka, koristio sam regresijski model jer predviđam kontinuiranu ciljnu varijablu (povrat).
Koristio sam LinearRegressor, RandomForestRegressor, XGBoost i LSTM, te sam pokušao i kombinaciju stackingom. Najbolji se pokazao RandomForestRegressor.

Performase sam procjenjivao s metrikama MAE, RMSE i R**2
Kako bi osiguraao da model nije overfitiran na povijesne podatke, korišten je time-based split gdje se treniranje vrši na starijim, a evaluacija na novijim uzorcima.
Dodatno, analizirane su performanse modela kroz različite vremenske periode (walk-forward validacija) i uspoređene metrike između trening i test skupa.
Budući da su performanse stabilne i nema značajnog pada točnosti, zaključuje se da model ne pokazuje znakove overfittinga.
U stvarnom vremenu model bi se mogao integrirati u sustav koji kontinuirano dohvaća nove OHLCTV podatke putem API-ja burze. Svaki novi zapis bio bi dodan u povijesni DataFrame a mobel bi odmah generirao predikciju sljedećeg satnog povrata.

3. API i kontejnerizacija

Nakon preuzimanja kreiranog modela, bilo je potrebno napraviti API. U tu svrhu sam koristio FastAPI. Kod je poprilično jednostavan. Postoje samo 3 endpointa:
/health – Koristi samo za provjeru stanja API-ja, napravljen više za moje potrebe testiranja.

/info – Vraća informacije o modelu za predviđanje povrata

/predict – Prima novi redak sirovih OHLCTV podataka i predviđa povrat.

Što se tiče ovog, koderskog, dijela zadatka, nije utrošeno previše vremena s obzirom da je potrebno samo napraviti 2 endpointa.

/info je jako jednostavan, na njegovo pozivanje se vraćaju informacije o korištenom modelu, podacima**,** značajkama, target koloni itd..

Za **/predict** pak je trebalo malo više truda i pokušaja kako bi radio kako treba.Osnovna premisa ovog endpointa je da prima sirove podatke od korisnika, računa značajke koje su korištene za treniranje modela i vraća predikciju povrata. Prvi problem s tim pristupom je bio što većina značajki zahtjeva podatke iz prošlih vremenskih perioda za računanje. Stoga je bilo potrebno učitati i historic_df, pa je za računanje .shift(n), .rolling(n), … korišteno n zadnjih redaka. Nakon što korisnik unese nove JSON podatke, dobiva povratnu informaciju o predviđenom povratu za taj period izražen u postotku. U slučaju pogrešno unesenih podataka, korisnik dobiva povratnu informaciju o tome što je pogrešno kako bi mogao ispraviti. Kod provjerava je li datum unesen u točnom formatu, je li datum prestar, pretvara vremenske zone, je li „close“ između „low“ i „high“ vrijednosti i još par sitnica. Pokrio sam sve osnovne stvari kako bi korisnik dobio točnu informaciju što da popravi, smatram da nema potrebe ulaziti u više mogućih grešaka.

Ako uneseni podaci prođu sve provjere, u historical_df se dodaje novi redak s podacima i značajkama. Model odabire samo posljednji redak i vrši predikciju. U slučaju nekih nepredviđenih grešaka sa strane API-a, vrši se logging i vraća se „internal server error“.

Posljednja stavka je bila kontejnerizacija. Docker je nešto s čim sam upoznat samo kroz samostalno istraživanje, nisam imao prilike raditi s njim u konkretnim slučajevima kroz školovanje i prijašnje pozicije, tako da se nadam da sam uradio sve što je trebalo. Uglavnom, kako ne bi došlo do situacija da se moj API ne može pokrenuti na računalu korisnika, npr. zbog nepoklapanja OS-a ili verzija neke biblioteke, vrši se kontejnerizacija. Model je spremljen u .pkl formatu pomoću joblib biblioteke. Nisam imao potrebe pisati .dockerignore.

Docker image se temelji na službenom „python:3.10-slim“ imageu. Nakon buildanja kontejnera, aplikacija se automatski pokreće pomoću uvicorn servera.