Analiza vremenskih serija u previđanju cijene dionica

Lakoseljac, Rafael Dominik

Sveučilište u Rijeci
Fakultet informatike
i digitalnih tehnologija

Sveučilišni diplomski studij Informatika

Rafael Dominik Lakoseljac

Analiza vremenskih serija u previđanju cijene dionica

Diplomski rad

Mentor: prof. dr. sc. Maja Matetić

Rijeka, kolovoz 2024.

Sveučilište u Rijeci
Fakultet informatike i digitalnih tehnologija

Rijeka, 27. svibanj 2024.

Zadatak za diplomski rad

Pristupnik: Rafael Dominik Lakoseljac

Naziv diplomskog rada: Analiza vremenskih serija u previđanju cijene dionica

Naziv diplomskog rada na eng. jeziku: Time series analysis in stock price forecasting

Sadržaj zadatka:

Zadatak diplomskog rada je primijeniti analizu vremenskih serija upotrebom različitih modela kao što su auto regresivni integrirani pomični prosjeci (ARIMA), Prophet i neuronske mreže. Potrebno je dati teorijski prikaz svakog modela te predstaviti rezultate dobivene njegovom primjenom. U radu je potrebno opisati proces prikupljanja podataka i njihovo pretprocesiranje. Dobivene rezultate primjenom različitih modela u zadatku predviđanja cijene dionica potrebno je usporediti i interpretirati.

Mentor/ica:
Prof. dr. sc. Maja Matetić

Voditeljica za diplomske radove:
Doc. dr. sc. Lucia Načinović Prskalo

Komentor/ica:

Zadatak preuzet: 16. travanj 2024.

(potpis pristupnika/ce)

Radmile Matejčić 2,
51000 Rijeka, Hrvatska

T: +385 (0)51/584-700
E: ured@inf.uniri.hr
www.inf.uniri.hr

OIB: 64218323816
IBAN: HR1524020061400006966

Sažetak

Rad se bavi analizom vremenskih serija u predviđanju kretanja cijena dionica pomoću različitih vrsta modela. Modeli su podijeljeni u četiri kategorije: Klasifikacijske, Auto regresivne integrirane pomične prosjeke (ARIMA), Prophet i Neuronske mreže. Pobliže je objašnjen princip rada svakoga modela te rezultati koji su se svakim od modela postigli. Uz opisivanje modela i njihovih rezultata, opisan je i sam proces prikupljanja podataka, njihove obrade i korištenja podataka u svakome od modela s ciljem dobivanja što boljega rezultata predviđanja. Svi dobiveni rezultati potom su uspoređeni s ciljem utvrđivanja koji je od njih dao najbolje rezultate, odnosno predviđanje. Na samome kraju prikazano je i verzioniranje modela kako bi se moglo pratiti rezultate modela, spremiti sam model radi buduće upotrebe i pratiti kako se sam model razvija i poboljšava u različitim verzijama i nadogradnjama.

Ključne riječi: vremenske serije, predviđanje, model, podaci, klasifikacija, neuronske mreže, treniranje, pretprocesiranje podataka.

Summary

The paper deals with the analysis of time series in predicting stock price movements using different types of models. The models are divided into four categories: Classification, AutoRegressive Integrated Moving Averages (ARIMA), Prophet, and Neural Networks. Each of these models is explained in detail, including the principles on which they operate and the results achieved with each model. Alongside the description of the models and their results, the paper also describes the process of data collection, processing and usage through each model to achieve the best possible prediction results. All obtained results are compared to determine which model provided the best outcomes or predictions. Finally, model versioning is presented to track model results, save the model for future use and monitor how the model develops and improves through different versions and upgrades.

Keywords: Time Series, Prediction, Model, data, Classification, Neural Networks, Training, Data Preprocessing.

Sadržaj

Uvod

U ovome radu bit će riječ o predviđanju cijene dionica američke tvrtke *International Business Machines* (IBM). Prikazat će se teorija i praksa analize vremenskih serija. Također, prikazat će se nekoliko različitih pristupa u rješavanju problema te će se pobliže objasniti i svaki od modela koji će služiti kao rješenje problema predviđanja cijena dionica.

Rad je strukturiran u devet glavnih poglavlja. U prvome je poglavlju opisana teorija vremenskih serija – u koje se svrhe sve mogu koristiti vremenske serije, objasnit će se neke od vrsta analiza vremenskih serija te varijance i klase podataka. U drugome poglavlju teorijski će se opisati svi modeli koji će se u radu koristiti kod predviđanja. Oni su podijeljeni u 4 kategorije: klasifikacijski model, auto regresivni integrirani pomični prosjeci, prophet i neuronske mreže. Trećim poglavljem započinje praktični dio rada te će se pobliže opisati podaci koje će se koristiti i akcije koje su bile potrebne kako bi se sami podaci mogli koristiti u radu. Četvrto poglavlje bit će vezano uz spremanje i dohvaćanje podataka iz baze podataka koja je postavljena. Sljedeća tri poglavlja prikazat će kako je teklo učenje modela te će se kratko predstaviti rezultati. Osmo poglavlje dat će dublji uvid u same rezultate koje su modeli postigli te prikazati prednosti i mane svakoga modela. Deveto poglavlje, ujedno i posljednje, prikazat će kako je moguće modele verzionirati pomoću vanjske aplikacije Mlflow. U zaključku rada bit će dana sinesteza čitavoga rada.

1. Vremenske serije

Podaci se u mnogim oblicima nalaze svuda oko nas. Mogu biti tekstovi, brojevi, valute, datum i vrijeme i mnogi drugi, a mogu dolaziti u mnogim formatima, primjerice slike, dokumenti, videozapisi. Ujedno, podaci mogu biti strukturirani i nestrukturirani. Vremenske serije specifičan su način prikupljanja, analize i zaključivanja nad podacima koji se prikupljaju tijekom nekoga vremenskog intervala. Osoba koja se bavi analizom takve vrste podataka same podatke prikuplja konstantno u nekom vremenskom intervalu, za razliku od prikupljanja podataka nasumično u nekom trenutku. Razliku koju ovakva vrsta analize pruža jest ta da se nad podacima prikaže koliko se različite varijable mogu mijenjati tijekom vremena. Za neke varijable vrijeme može predstavljati krucijalnu ulogu zbog toga što je moguće prikazati kako se sam podatak može mijenjati u postavljenome periodu promatranja. Samim time takva analiza može proizvesti mnogo drukčije rezultate jer se u samo promatranje dodaje još jedan izvor informacija koji daje poveznice između podataka i vremena promatranja. Vremenske serije za analizu moraju koristiti veliku količinu podataka, odnosno podatkovnih točaka u kojima se bilježe promatrane varijable. Velika količina podataka kod analize pomaže smanjiti ili u potpunosti izbaciti šum u promatranju, ujedno osigurava hvatanje svih trendova i obrazaca. Trendove ili obrasce potrebno je promotriti i analizirati kako bi se zaključilo da nisu ekstremi te da mogu objasniti sezonsku varijaciju. Količina podataka koja se prikupi tijekom analize te sama krajnja analiza podataka pomoću vremenskih serija može se koristiti kod prognoziranja budućih događaja, bile to vremenske prilike ili neprilike, cijene dionica i mnoge druge vrste podataka nad kojima je vrijeme jedan, ako ne i najvažniji, čimbenik.

1.1. Korištenje vremenskih analiza u poslovnim sustavima

Vremenske analize pomogle su mnogim tvrtkama u nošenju s kriznim stanjima, prekomjernom proizvodnjom proizvoda, smanjenjem gubitaka i, najvažnije, povećanjem prihoda. Postupci analize podataka mogu tvrtkama ukazati trendove na tržištu kao i sezonalnost njihovih proizvoda. Samom vizualizacijom podatka može se prikazati trendove i sezonalnosti na grafovima te ih usporediti s nekim drugim podacima koji su se dogodili u vrijeme kada je trend nastao ili kada je trend počeo opadati. Naravno, kako bi se takva predviđanja mogla analizirati, potrebno je mnogo podataka i vremena, ali u današnje je vrijeme do podataka mnogo lakše doći nego prije, što je za tvrtke vrlo dobro, ali ne i za krajnjega korisnika jer se njegovi podaci prikupljaju na društvenim mrežama i mnogim drugim mjestima te se prodaju tvrtkama

kako bi se određeni proizvod mogao reklamirati i njegova prodaja povećati. U poslovne svrhe moguće je koristiti vremenske serije za mnoge primjene. Neke od njih bit će prikazane u nastavku.

1.2. Primjeri korištenja vremenskih serija

Dotakli smo se primjene analize vremenskih serija u poslovne svrhe, koje su vjerojatno i najkorištenije, ali uz te svrhe postoje još mnoge druge. Prikupljanje podataka o vremenu jedna je od čestih primjena zbog same raznovrsnosti te je moguće analizirati temperature, padaline i mnoge druge aspekte vremenske prognoze. Vremenske serije mogu se koristiti i u medicinske svrhe kod očitavanja rada srca (EKG) i očitavanja rada mozga (EEG). Uz korištenje u poslovne svrhe kod prodaje proizvoda, ujedno je široko rasprostranjeno i korištenje vremenskih serija u svrhe predviđanja cijena dionica, kamata i za druge financijske pojave koje se s vremenom mogu mijenjati i kod kojih je vrijeme ključan čimbenik. Svi ovi primjeri relativno su lako dostupni i u kratkom se roku može prikupiti vrlo velika količina podataka. Kod nekih je analiza lakša, ali kod nekih je potrebno uspoređivati još neke podatke koji nisu dostupni u samom cilju analize vremenskih serija.

Jedan od takvih slučajeva jest predviđanje cijene dionica o kojima će i biti riječ u ovome radu. Na samim podacima lako je zaključivati i predviđati, ali nikada se ne može sa 100 % sigurnošću znati kojim će smjerom određena dionica otići zbog drugih faktora poput ratova ili nekih drugih vanjskih čimbenika koji nisu u okviru predviđanja, ali imaju utjecaj na podatke. Poznati slučaj utjecaja vanjskih čimbenika na cijenu dionica je slom burze iz 1929. godine kada se velika količina dionica kupovala zbog ekspanzije tržišta dionica te su se ljudi oglušili na moguće rizike. Milijuni dolara „izlili“ su se iz banaka, ušteđevina i raznih izvora samo kako bi se kupile dionice. Nakon toga, kada su cijene počele opadati, interes za prodaju bio je mnogo veći nego kod kupnje te je samim time cijena dionica počela padati u obliku slobodnoga pada i izazvala slom burze koji je nastupio 18. listopada 1929. godine. Samo 5 dana nakon toga, 24. listopada, na dan poznat kao „Crni Četvrtak“, prodano je 12,9 milijuna dionica u jednom danu jer su investitori pokušavali pokriti svoje gubitke. Taj događaj u slučaju vremenskih serija nije lako predvidjeti jer se takvi skokovi ne događaju često pa njihovo ponašanje nije moguće dobiti iz samih podataka koji su vezani uz cijenu dionica.

1.3. Vrste analiza vremenskih serija

Analize vremenskih serija mogu uključivati mnoge kategorije ili varijacije podataka pa su zbog toga analitičari prisiljeni izrađivati kompleksnije modele. Analitičari kod izrade modela ne mogu uzeti u obzir sve varijance podataka koje imaju te ujedno ne mogu generalizirati model za svaki uzorak podataka na koji naiđu. Kompleksni modeli koji pokušavaju odraditi mnoge stvari dolaze do problema da ne pristaju (engl. *fit*) na podatke na kojima se analizira. Nepristajanje (engl. *fitting*) nad podatcima ili pretjerano pristajanje (engl. *overfitting*) nad podacima može uzrokovati da naš model ne razlikuje slučajnu pogrešku od pravih odnosa između podataka. Samim time naša analiza gubi smisao jer su rezultati netočni ili iskrivljeni.

1.3.1. Klasifikacija

Klasifikacija, kako i samo ime kaže, govori o određenim klasama, u slučaju s vremenskim serijama podacima se dodaje određena klasa ovisno o našoj primjeni i podacima. U nastavku ovoga rada bit će prikazan jedan primjer klasifikacije nad cijenama dionica i u tom će slučaju klasa koja će biti dodijeljena govoriti o tome hoće li cijena u sljedećem ciklusu promatranja padati ili rasti. Ovo je samo od jedan primjer klasifikacije, ali u praksi je takva vrsta lako primjenjiva na velik broj slučajeva, primjerice padaline.

Modeli klasifikacije rade na principu računanja udaljenosti. Što znači da kada uspoređujemo dva objekta promatranja, što je njihova udaljenost manja, to su ti objekti sličniji i pripadaju istoj klasi. Pripadnost klasi određuje se prema unaprijed zadanim vrijednostima. Takve se udaljenosti najčešće računaju jednom od sljedećih metoda: *Euclidian, Hamming, Manhattan* ili *Minkowski* metoda udaljenosti. Dobiveni rezultati koriste se u paru s nekim od poznatih algoritama koji rade s udaljenostima, kao npr. KNN (engl. *k-nearest neighbors*). Taj algoritam radi na vrlo jednostavnom principu koji uzima jedan skup podataka koji potom dijeli na dva manja – jedan skup za učenje i jedan skup za testiranje. Nakon toga na podacima za učenje istrenira sam model. Model uzima udaljenosti koje je dobio i uspoređuje ih s podacima iz skupa za testiranje i na temelju udaljenosti tih podataka radi zaključke kojem skupu pripadaju podaci za testiranje. Još jedan od poznatijih algoritama je SVM (engl. *support vector machine*). Ovaj algoritam, isto kao i KNN, koristi izračunate udaljenosti kako bi donosio odluke, ali u ovom algoritmu koriste se hiperravnine ili linije s dvije dimenzije. Isto kao i kod prošloga modela uzimaju se u obzir udaljenosti, samo što se u ovom slučaju uzima udaljenost hiperravnine od podatka za učenje i podatka za testiranje te se na temelju toga dodjeljuju klase.

1.3.2. Segmentacija

Segmentacija jest strategija analize vremenskih serija gdje se sami podaci dijele tako da se kod dijeljenja uzima u obzir vrijeme i tako se kreiraju segmenti naših podatka. Cilj ovakve analize je da se iz podataka izvuku uzorci tako da se promatraju karakteristike svakoga segmenta, a na kraju i same cjeline podatka. Kod segmentacije imamo tri glavna pristupa za podjelu skupa, a to su: *Top-down, Bottom-up* i *Sliding window*. *Top-down* ili dosl. *s vrha prema dnu* jest metoda podjele kod koje se radi na principu „podjeli pa vladaj“ te tako uzima cijeli skup podataka i od toga skupa radi dva segmenta koje podjeli tako da su sami segmenti što različitiji, a onda se taj proces ponavlja sve dok se ne otkrije nekakav uzorak. *Bottom-up* ili dosl. *odozdo prema gore* jest metoda koja na samom početku naše podatke dijeli na više segmenata te se potom za svaki od tih segmenata računa funkcija pogreške koja predstavlja razliku između stvarnih podataka i podataka koje naš segment predstavlja. Cilj je ove metode pronaći segmente s najmanjom funkcijom pogreške te se potom ti segmenti međusobno spajaju na principu gdje se spajaju segmenti s najmanjim funkcijama pogreške. Taj se proces provodi dok se ne postigne ciljani rezultat funkcije pogreške ili dok se svi segmenti ne spoje. *Sliding window* ili dosl. *klizeći prozor* jest metoda koja kreira „prozor“, npr. koji definira početak na lijevom rubu i kraj na desnom rubu podatka. Kod ovoga modela prozor se pomiče u inkrementima prema desno dok se ne dođe do krajnje linije koju je postavio promatrač. Aplikacije segmentacije mogu se koristiti kod analize trendova, predviđanja budućih događaja, smanjivanja buke i detekcija anomalija u podacima.

1.3.3. Predviđanje

Previđanje jest strategija analize podatka koja sa što većom preciznošću pokušava predvidjeti buduće događaje, odnosno buduće podatke. Takva metoda zasniva se na prikupljanju povijesnih podatka koji će služiti kao materijal kod izrade modela i na temelju zaključivanja nad tim podacima predvidjeti buduće događaje. Kod predviđanja nije moguće sa 100 % preciznošću predvidjeti buduće događaje, ali je moguće predvidjeti događaje koji imaju veće šanse, odnosno događaje koji imaju manje šanse da se dogode. Bolje rezultate predviđanja najčešće ćemo dobiti ako baratamo većom količinom podataka i što kvalitetnijim podacima te njima gradimo model. Kod analize vremenskih serija predviđanje ne služi samo kako bi se predvidio budući događaj nego i kako bi se dobilo shvaćanje što je uzrokovalo taj budući događaj, odnosno što stoji iza toga predviđanja.

U ovome radu naglasak je na predviđanju cijena dionica o kojima će se kasnije detaljnije govoriti, ali neke od metoda koje su korištene u radu su LSTM (engl. *Long Short Term Memory*), CNN (engl. *Convolutional Neural Network*) i GRU (engl. *Gated Reccurent Unit*). Naravno, navedeni modeli ne služe samo u svrhu predviđanja, ali u ovom radu korišteni su u tu svrhu, no više o tome bit će rečeno nešto kasnije.

Kao što je i prethodno navedeno, predviđanje ima mnoge primjene u stvarnom svijetu kod prognoze, zdravstvenih nalaza, ekonomskih rezultata i mnogih drugih. Razlog tomu je taj da bi svi ljudi voljeli znati što je u budućnosti, ali za predviđanje budućnosti potrebna je velika količina podataka koji su relevantni za naš specifičan slučaj. Naravno, i u slučaju gdje imamo veliku količinu podatka i vrlo relevantne podatke nikada nećemo sa 100 % sigurnošću moći reći da će se taj događaj dogoditi jer uvijek postoje anomalije koje nisu toliko česte da bi se mogle predvidjeti ili očekivati.

1.4. Varijacije podataka i klase podataka

Podatke kod vremenskih serija možemo podijeliti na dvije glavne klase, a to su: *Stock time series data* i *Flow time series data*. *Stock time series data* ili dosl. *zalihe vremenskih podataka* jesu klasa podataka koja nastaje kada uzmemo „sliku“ podataka, odnosno kad uzmemo podatke u nekom intervalu koji nije promjenjiv. *Flow time series data* ili dosl. *dotok vremenskih podataka* jest kada konstantno dobivamo podatke i tijekom analize proučavamo sve varijable nad tim podacima. Varijacije podataka mogu se koristiti kod tri glavne vrste analize, a to su: 1. funkcionalna analiza, 2. analiza trendova i 3. analiza sezonalnosti.

Funkcionalna analiza jest analiza u kojoj je moguće razaznati uzorke i odnose u samim podacima te na temelju tih zapažanja identificirati bitne događaje koji su ti uzorci uzrokovali.

Analiza trendova bila je ranije spomenuta te ona spada u varijaciju podataka jer se na temelju te analize može odrediti određeno kretanje podataka i što ih uzrokuje. Postoje dvije vrste: deterministička i stohastička. Deterministička analiza trendova jest analiza kod koje je moguće pronaći uzrok koji je naveo na taj trend, a stohastička analiza podatka nepredvidiva je i slučajna.

Sezonska varijacija opisuje događaje koji se događaju uvijek i u specifično vrijeme u intervalu promatranja.

2. Vrste modela koje će se koristiti u radu

U prijašnjim poglavljima bila je riječ o samoj vremenskoj analizi, vrsti analiza podataka te varijaciji i klasama podataka. U ovome će poglavlju biti riječ o vrstama modela i modelima koji će se koristiti. Vrste modela možemo podijeliti u četiri kategorije: 1. klasifikacijski modeli, 2. auto regresivni integrirani pomični prosjeci, 3. Prophet model za predviđanje i 4. neuronske mreže. Svaka će kategorija u daljnjim dijelovima rada biti pobliže opisana te ujedno i različiti modeli koji spadaju u te vrste modela.

2.1. Klasifikacijski modeli

Klasifikacija spada pod nadzirani oblik strojnoga učenja gdje model koji se kreira i trenira pokušava predvidjeti u koju skupinu ili klasu pripadaju ulazni podaci. Modeli u klasifikaciji treniraju se na podacima za učenje te se sama preciznost i korisnost modela vrjednuje na podacima za testiranje. Kasnije u radu bit će prikazana podjela podataka, ali generalno se uzima neki problem te se uzimaju podaci nad kojima se želi riješiti određeni problem. Te podatke dijelimo na dvije skupine: 1. podatke za učenje, koji najčešće iznose između 70 i 80 % ukupnih podataka i 2. podatke za testiranje, koji iznose ostatak podataka koji nisu ušli u podatke za učenje. Uz te dvije skupine moguće je dodati i treću skupinu podataka, a to je skupina podataka za validaciju koji služe za namještanje modela s ciljem dobivanja boljih rezultata od samoga modela.

Klasifikacijske modele možemo podijeliti u dvije skupine „Lijeni učenici" i „Željni učenici". Željni učenici (engl. *Eager Learners*) su algoritmi koje sam model najprije izgradi na podacima za učenje prije davanja predviđanja ili zaključaka na budućim podacima. Takvi algoritmi provode više vremena u građenju i treniranju samoga modela, kako bi mogli naučiti „težinu" svake varijable podatka te je ujedno takvim algoritmima manje vremena potrebno za donošenje zaključaka. Najčešći takvi modeli su: logistička regresija, drvo odluke i SVM (*Support Vector Machine*). Lijeni učenici funkcioniraju na način da ne kreiraju model na podacima za učenje, već svakoga puta kada je potrebno neko predviđanje pretražuju sve od prije zapamćene podatke kako bi pronašli najbližega susjeda našim ulaznim podacima. Upravo je taj proces zaslužan za naziv ove skupine algoritama. Navedeni su algoritmi lijeni zbog same činjenice da se ne gradi model. Zbog takvoga načina predviđanja gdje je svaki put potrebno učitati sve podatke, navedeni algoritmi su spori i što je veća količina podataka, taj je model sporiji. Neki od tih algoritama su: K-najbliži susjedi i rezoniranje temeljeno na slučaju (engl. *Case-based*

reasoning). Klasifikacijski modeli mogu imati različite zadatke, npr. binarna klasifikacija, klasifikacija s više klasa ili klasifikacija s više oznaka.

Binarna klasifikacija, kao što i samo ime govori, je model koji predviđa jednu od dviju klasa nad ulaznim podacima. Takvi modeli koriste se kod detekcije neželjene elektroničke pošte i sličnih situacija gdje imamo samo dvije klase. Mnogi od prije spomenutih modela mogu se koristiti kod izgradnje binarne klasifikacije. Jedan od takvih modela korišten je i u ovom radu te će o samom modelu biti riječ kasnije.

Klasifikacija s više klasa slična je binarnoj klasifikaciji, samo što u ovom slučaju imamo više od dviju klasa. Većina algoritama koji se koriste za binarnu klasifikaciju mogu se koristiti i u slučaju s više klasa. Kod binarne klasifikacije uspoređujemo podatke na temelju toga koliko je ulazni podatak sličan jednoj ili drugoj klasi, odnosno model se trenira tako da uzima podatke za učenje i trenira tako da klasificira podatke u jednu klasu pa u drugu te tako uspoređuje nove ulazne podatke koji su sličniji prvoj ili drugoj klasi. U slučaju kada imamo više od dviju klasa uzimamo sve klase koje imamo te ih smatramo zasebnim entitetima koji dijele oznaku. Takve entitete uspoređujemo na način da uspoređujemo jednu klasu sa svima ostalima i tako radimo za sve ulazne podatke. Time klasifikacija s više klasa postaje binarna klasifikacija jer se uspoređuje jedna klasa sa svima ostalima, što čini samo dvije klase. Time se smanjuje vrijeme potrebno kod klasifikacije i treniranja.

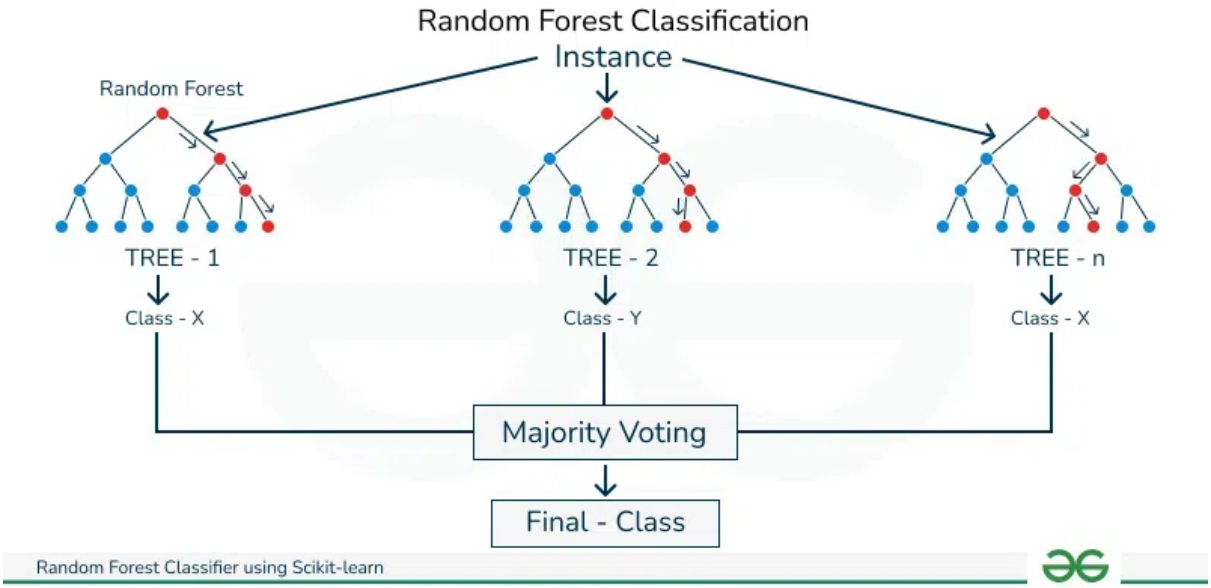
Klasifikacija s više oznaka slična je, a ujedno i različita od klasifikacije s više klasa iz razloga što u ovoj klasifikaciji naš model pokušava predvidjeti za svaki ulazni podatak pripada li on 0 ili više klasa. Takva vrsta klasifikacije može se koristiti za prepoznavanje objekata na slici, npr. model koji predviđa koja vrsta životinje je na slici – pas, mačka, konj i slično. Modeli koji se koriste za binarnu klasifikaciju ili više klasnu klasifikaciju ne mogu se koristiti za ovu vrstu klasifikacije, ali mnogi algoritmi imaju svoju specijaliziranu vrstu koju koriste kod klasifikacije s više oznaka. Problemi koji se mogu javljati kod klasifikacija jest neravnomjerna distribucija podataka. Neravnomjerna distribucija podataka može biti zaslužna za loše rezultate klasifikacije. Razlog tomu je što u slučaju kad imamo tri klase i te su klase neravnomjerno raspoređene, model automatski uči biti pristran (engl. *biased*) prema klasi koja ima najveći postotak podataka od ukupne količine podataka. To predstavlja velik problem u situacijama gdje su rijetki slučajevi vrlo važni podaci, npr. prognoza rijetke bolesti ili prepoznavanje lažnih transakcija i sličnih slučajeva gdje se želi otkriti nešto o čemu imamo mnogo manju količinu podataka, nego kod uobičajenih podataka. Naravno, takvim klasama podatka potrebno je promijeniti distribuciju kako bi se nad podacima i dalje moglo provoditi istraživanje i učenje

modela. Neke od tih metoda su: nasumična eliminacija primjera iz većinske klase ili nasumično ponavljanje primjera iz manjinske klase. Takvim metodama moguće je ujednačiti većinsku i manjinsku klasu kako bismo istrenirali model koji će moći dati precizne rezultate i precizna predviđanja.

2.1.1. Slučajna šuma (engl. *Random forest)*

Slučajna šuma (engl. *Random forest*) klasifikacijski je model koji će se koristiti u ovome radu te će kasnije biti prikazano kako je teklo njegovo učenje i rezultati koji su dobiveni. Sada će pobliže biti riječ o samom algoritmu i njegovom teorijskom načinu funkcioniranja.

Slučajna šuma ili slučajna šuma odluke nadgledani je model strojnoga učenja koji se koristi za klasifikaciju, regresiju i ostale slučajeve koji se mogu riješiti pomoću stabala odlučivanja (engl. *Decision trees*). Takav algoritam pogodan je za veliku količinu podataka i podatke velike kompleksnosti, gdje sami podaci imaju više dimenzija i gdje je potrebno imati uvid u sve značajke podataka koji se proučavaju. Mogućnosti ovoga modela zaslužne su da model ostaje precizan, a minimizira se pretreniranost. Nasumične šume kreiraju setove stabala odlučivanja koji se kreiraju tako da se uzmu nasumični podskupovi (engl. *Subset*) podataka iz cjelokupnoga seta podataka. Kada se dobio željeni broj takvih stabala, provodi se glasanje koje je stablo najbolje rješenje.



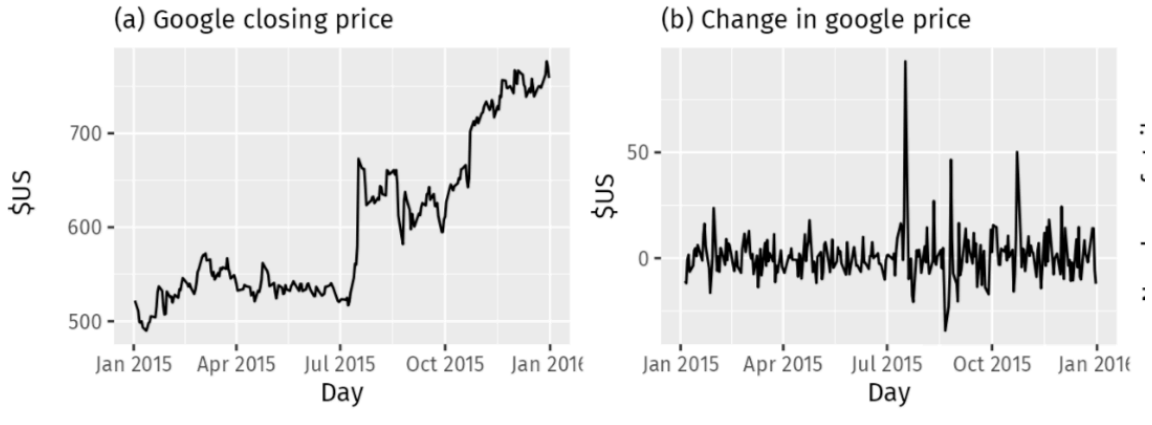
Slika 1. *Prikaz stabala odluka*

Tijekom faze učenja koristi se metoda pakiranja (engl. *bagging*) koja pomaže modelu da ne trenirana nad dominantnim značajkama te da time postane pristran dominantnoj značajki, već da bude diversificiran, ne pristran i da podjednak broj značajki bude u svim stablima odlučivanja.

2.2. Auto regresivni integrirani pomični prosjeci

Auto regresivni integrirani pomični prosjeci (engl. *Autoregressive Integrated Moving Average – ARIMA*) uz eksponencijalno izglađivanje jedni su od dva najčešće korištena pristupa predviđanja vremenskih nizova i pružaju komplementarne pristupe problemima. Modeli eksponencijalnoga izglađivanja temelje se na opisu sezonalnosti nad podacima. ARIMA modeli stoga imaju cilj opisati autokorelaciju nad podacima. Kako bismo mogli opisati ARIMA modele, najprije je potrebno raspraviti o dvama konceptima, a to su koncept stacionaranosti i koncept razlikovanja.

Stacionarne vremenske serije su vremenske serije kojima statističke značajke ne ovise o vremenu u kojem se promatraju. Samim time vremenske serije sa sezonalnostima i trendovima ne spadaju u tu kategoriju. Razlikovane vremenske serije (engl. *Differencig time series*) su serije u kojima vrijeme promatranja igra ulogu u otkrivanju trendova i sezonalnosti, ali pomoću transformacija moguće je stabilizirati varijaciju u vremenskim podacima.



Slika 2. *Transformacija varijance*

Slika 2. prikazuje baš takvu transformaciju. Na lijevoj slici, ili na slici (a), prikazana je cijena zatvaranja dionica Googlea, a na desnoj slici, ili na slici (b), prikazana je promjena u cijenama Googlea. Slika (a) prikazuje kako podaci nisu stacionarni i variraju, a slika (b) kako su podaci stacionarni s blagim odstupanjima. Transformacijom je moguće stabilizirati varijacije u

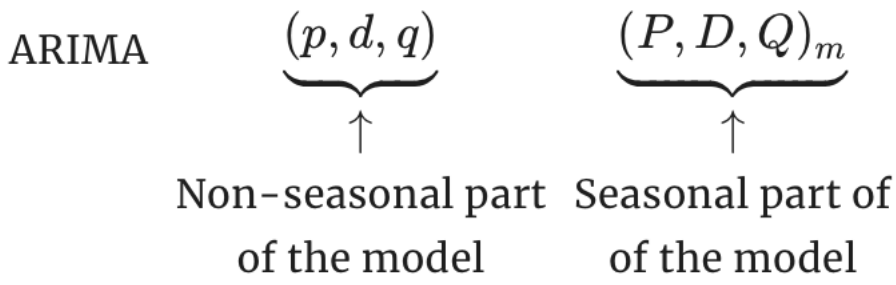
vremenskim serijama te samim time možemo stabilizirati učinak vremena i ukloniti, ili uvelike smanjiti, trendove ili sezonalnosti. U auto regresivnom modelu predviđamo varijablu koja nas interesira na način da koristimo linearnu kombinaciju prošlih varijabli. Sam izraz auto regresija govori nam o tome da se radi o regresiji varijable u odnosu na samu sebe. Auto regresivni modeli su vrlo fleksibilni kod rukovanja sa širokim spektrom različitih uzorka u vremenskim serijama. Auto regresivni integrirani pomični prosjeci rade u kombinaciji auto regresivnoga modela i modela pomičnoga prosjeka. Model pomičnoga prosjeka je model koji ne uzima prošle vrijednosti varijable, već koristi pogreške koje su se dogodile kod prošlih predviđanja. Kada spojimo te dvije vrste modela, dobivamo ne sezonalni ARIMA model. U nastavku ovoga rada opisat će se kako rade ARIMA i Sezonalni auto regresivni integrirani pomični prosjeci (engl. *Seasonal Autoregressive Integrated Moving Average* – S*ARIMA*) modeli, a iduća poglavlja bave se teorijskim aspektom tih modela.

2.2.1. ARIMA

ARIMA model sastoji se od 3 dijela AR(p) broj auto regresivnih članova, I(d) razlika u ne sezonskim opažanjima i MA(q) veličina prozora pomičnoga prosjeka. Uz sve dijelove prikazana su i slova (p, d, q) koja se kod treniranja modela postavljaju na nama željene parametre veličina kako bismo naš model mogli što bolje namjestiti da dobijemo bolje rezultate, ujedno vrijednosti 0 su isto prihvatljive. ARIMA model može se izgraditi pomoću sezonalnih ili ne sezonalnih formata podataka, ali te podatke pretvara u stacionarne, odnosno ako koristi sezonalne podatke, tu sezonalnost miče i podaci postaju ne sezonalni.

2.2.2. SARIMA

SARIMA model je nadogradnja ARIMA modela te slovo S označava sezonalnost. Odnosno u klasičan ARIMA model dodaje se sezonalnost u ne sezonalne komponente. ARIMA modeli široko se koriste kod vremenskih serija i predviđanja, dok su SARIMA modeli specijalno dizajnirani kako bi rukovali sa sezonalnim uzorcima.



Slika 3. *ARIMA i SARIMA formule*

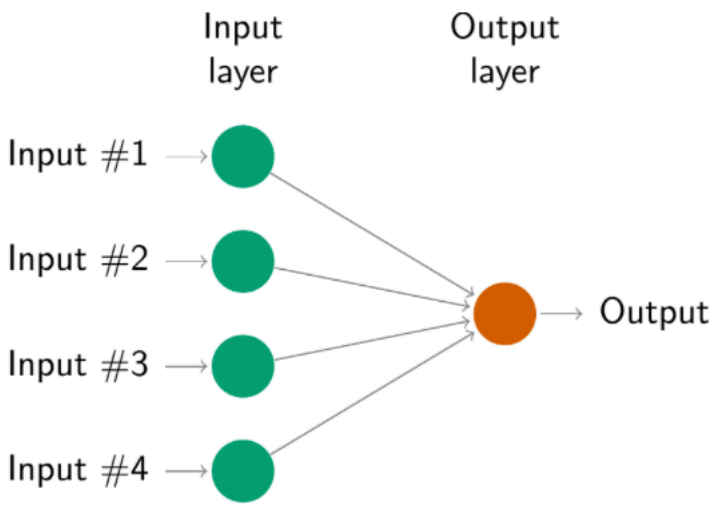
Na Slici 3. prikazan je model ARIMA s lijeve strane i SARIMA s desne, a razlika u modelima je dodatak slova *m* kod SARIMA modela. Slovo *m* označava broj zapažanja u jednoj godini. Ujedno kod SARIMA modela slova (P, D, Q) označena su velikim tiskanim slovima jer se u ovom slučaju radi o sezonalnim podacima. Slovo P je sezonalna auto regresija, slovo D je sezonalna razlika koja u obzir uzima veličinu koja je potrebna da bi se maknula sezonalnost iz vremenske serije i slovo Q je sezonalni pomični prosjek.

2.3. Prophet

Prophet je procedura za predviđanje vremenskih serija i temelji se na principu aditivnoga modela, gdje se nelinearni trendovi dodaju s godišnjom, tjednom i dnevnom sezonalnosti uz dodatak utjecaja blagdana. Sam model najbolje radi s podacima koji imaju vrlo velik utjecaj sezonalnosti. Model je vrlo otporan na nedostatak podataka i tipično vrlo dobro podnosi granične podatke. Prophet je *open source* softver koji je razvio Facebook te je dostupan za preuzimanje na sljedećoj poveznici: https://pypi.org/project/prophet/. Model se koristi u mnoge svrhe u cijeloj Facebook aplikaciji kako bi se mogli dobivati pouzdani i točni podaci. Time se ostvaruje dobro planiranje i postizanje ciljeva koje je tvrtka zadala. Facebook Prophet koristi automatski, to znači da je model prepušten sam sebi te ga se puni novim podacima. Prophet se, kao što je prethodno navedeno, dobro nosi s nedostatkom podataka, rubnim podacima te drastičnim promjenama koje mogu nastati u vremenskim serijama. Omogućava mnoge mogućnosti kod predviđanja koje korisnik može svojim znanjem o domeni postaviti u svoju korist kako bi predviđanje dalo bolje rezultate za određene domene primjene. Prophet je dostupan u R jeziku i Pythonu.

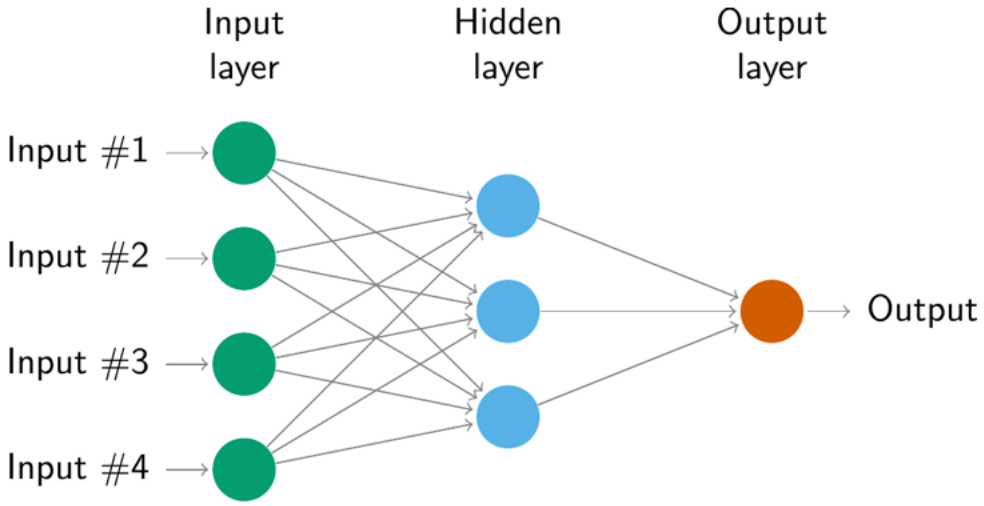
2.4. Neuronske mreže

Neuronske mreže umjetno su kreirane metode predviđanja koje se baziraju na matematičkom modelu mozga. Samim time što je to „mozak“ modelu omogućava da radi kompleksna predviđanja koja imaju povezanost između nelinearnih varijabli i njegovih prediktora. Neuronska mreža može se smatrati mrežom koja je sastavljena od neurona koji su organizirani u slojeve. Ulazni podaci ili prediktori su donji slojevi, a izlazni podaci ili predikcije formiraju gornji sloj. Uz ulazne i izlazne slojeve mreže mogu imati i slojeve između, koji se nazivaju sakriveni slojevi ili sakriveni neuroni.



Slika 4. *Jednostavan model neuronske mreže*

Slika 4. prikazuje jednostavan model neuronske mreže koja se sastoji od ulaznoga sloja s lijeve strane koji čine četiri ulaza neurona ili četiri prediktora te izlaznoga sloja koji čini jedan izlazni neuron ili jedna predikcija. Koeficijenti pridruženi prediktorima nazivaju se još i ponderi ili težine. Težine se odabiru pomoću algoritma učenja koji minimizira funkciju troškova u okviru same neuronske mreže. U jednostavnom primjeru, kao što je prikazan na Slici 4., može se koristiti i linearna regresija koja je mnogo učinkovitija metoda treniranja jednostavnijega modela.

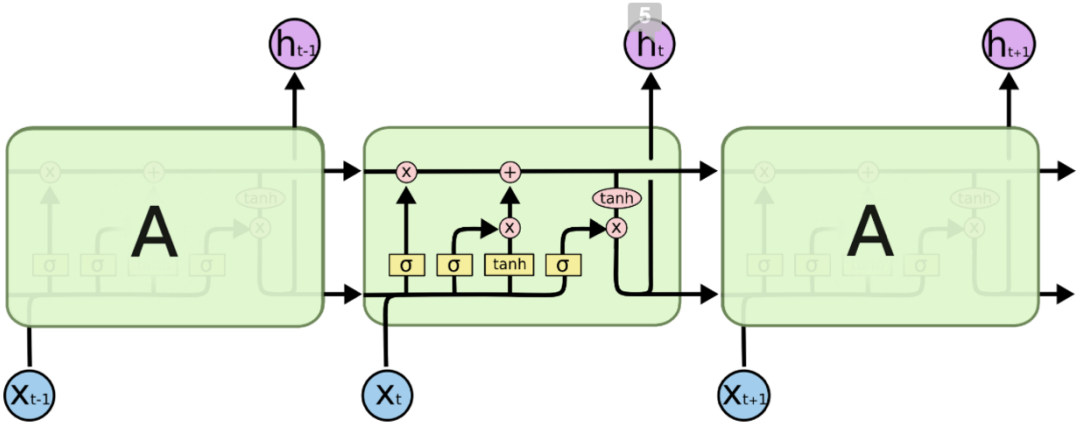


Slika 5. *Model neuronske mreže sa sakrivenim slojem*

Na Slici 5. prikazana je malo kompleksnija mreža sa sakrivenim slojem koji čine tri neurona. Ovakav model još se može i nazvati višeslojna *feed-forwaed* mreža. Izlaz iz svakoga sloja ulaz je u sljedeći sloj. Ulazi se kombiniraju pomoću ponderirane linearne kombinacije. Rezultat se prije samoga izlaza modificira nelinearnom funkcijom. Vrijednosti težina se ograničavaju kako ne bi postali preveliki. Parametar koji ograničava težine da postanu prevelike naziva se i parametar raspadanja, koji se najčešće postavlja na 0,1. Težine se na početku treniranja nasumično odabiru, a nakon toga se mijenjaju pomoću promatranih podataka. Zbog toga elementa nasumičnosti u neuronskim mrežama sam model se trenira nekoliko puta kako bi se postigao bolji rezultat predviđanja. To se radi na način da se mijenjaju početne točke te se na kraju radi prosjek svih rezultata. Broj sakrivenih slojeva i broj čvorova bitno je specificirati unaprijed. Oni se obično odabiru pomoću unakrsne validacije.

2.4.1. LSTM

LSTM (engl. *Long Short Term Memory*) posebna je vrsta RNN-a (rekurentne neutralne mreže, engl. *Recurent Neutral Network*) koja može učiti dugoročne odnose između podataka. Ove mreže kreirale su se u svrhu borbe s problemom dugoročnosti. LSTM-u je normalno da pamte veliku količinu dugoročnih odnosa kako bi pomoću toga mogli donositi predviđanja.

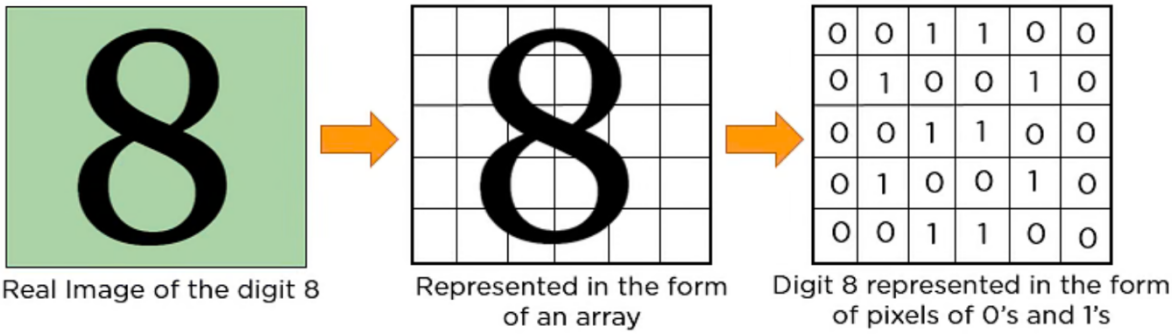


Slika 6. *LSTM model*

Slika 6. prikazuje pojednostavljeni prikaz jednoga LSTM-a, a sada ćemo pobliže proučiti kako on funkcionira. LSTM radi poput pokretne trake na kojoj se oduzimaju i dodaju informacije kroz tzv. prolaze (engl. *gates*). Usredotočimo se na središnji dio slike koji prikazuje sam algoritam. Na vrhu srednjega pravokutnika prikazana je linija koja čini pokretnu liniju, na nju se dodaju i oduzimaju podaci. Prvi ulaz u tu liniju s lijeve strane je sigmoidni sloj koji je zaslužan za količinu podatka koja će se spremiti i može poprimiti vrijednost od 0 do 1. Ako je postavljen na 0, ne propušta nijedan podatak, a ako je postavljen na 1, propušta sve podatke. Sljedeći korak algoritma sastoji se od dva sloja – jedan je opet sigmoidni sloj koji odlučuje koje vrijednosti će se ažurirati, a drugi sloj je tanh sloj koji kreira vektore za nove vrijednosti. Potom se ta dva sloja spoje i vrijednosti šalju dalje kroz model. Na kraju dolazimo do izlaznoga sloja modela koji se opet sastoji od sigmoidnog sloja koji filtrira koje dijelove ćemo propustiti kao rezultat. Nakon toga taj filtrirani dio prolazi kroz tanh kako bi se vrijednosti postavile na vrijednosti između -1 i 1 te se množe s izlazom iz sigmoidnoga sloja, kako bismo odlučili koje ćemo podatke prikazati. Ovo je samo jedna od mnogih varijanti kako se LSTM može postaviti.

2.4.2. CNN

CNN ili konvolucijska neuronska mreža (engl. *Convolutional Neural Network*) je *feed-forward* neuronska mreža koja se tipično koristi za analiziranje slika pomoću topologije oblika rešetke.

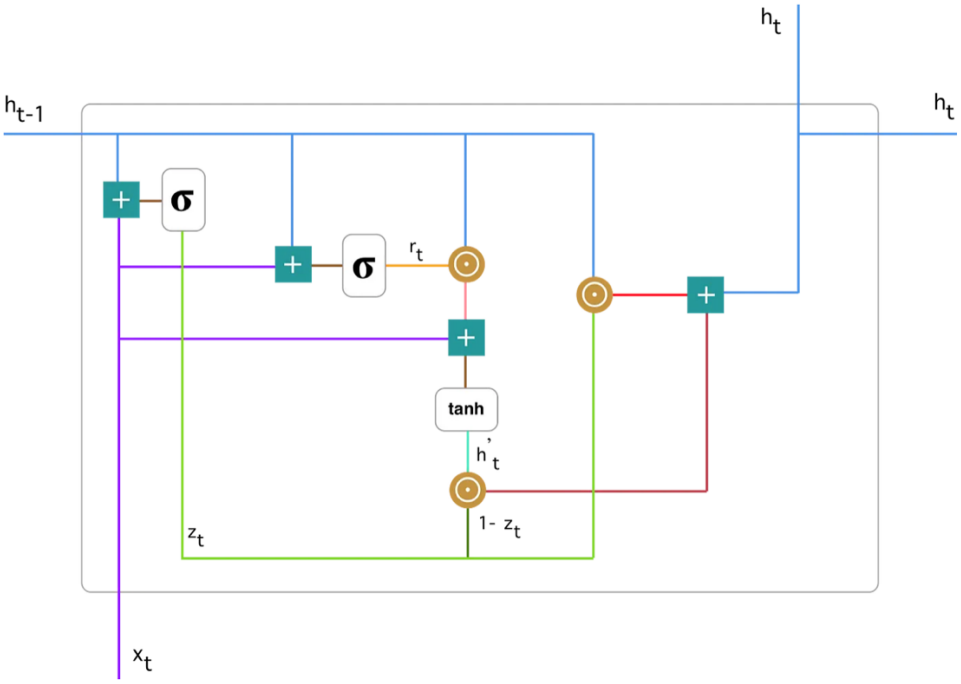


Slika 7. *Pretvaranje broja 8 u rešetku pomoću CNN-a*

Na Slici 7. prikazana je slika broja 8 koji se pomoću CNN-a pretvara u rešetku i sam CNN na zadnjoj slici prepoznaje gdje se pikseli na slici nalaze. Na mjestima gdje se nalazi piksel dodjeljuje se broj 1, a gdje nema piksela 0. Na temelju toga radi se predviđanje o kojem se broju radi. Konvolucijska neuronska mreže ima mnoge sakrivene slojeve kako bi se iz slike moglo izvući predviđanje. Četiri glavna sloja su konvolucijski sloj, ReLu sloj, *Pooling* sloj i potpuno povezani sloj. Konvolucijski sloj prvi je sloj u redu CNN-a, uzmimo primjer slike iznad. Imamo matricu veličine 5x6, konvolucijski sloj određuje matricu koja će se koristiti kako bi se smanjila prva matrica u npr. matricu veličine 2x2 koja služi kako bi se izračunao engl. *dot produkt* ili točkasti produkt kako bi se dobila konvolucijska matrica značajki. ReLu sloj je kratica za ispravljenu linearnu jedinicu. Nakon što se kreirala matrica značajki, ReLu sloj izvodi operacije nad tim elementima matrice i postavlja negativne piksele na 0 te unosi nelinearnost u mrežu i generira izlaz ispravljenih značajki. *Pooling* sloj dolazi sljedeći i koristi se kako bi se smanjila dimenzionalnost značajki iz prijašnjega sloja tako da uzima filter kao kod konvolucijskoga sloja, ali u ovom sloju u tom filteru veličine npr. 2x2 uzima se najveći broj iz toga filtera i sprema se samo njegova vrijednost za daljnji sloj. Nakon toga provodi se izravnavanje. Izravnavaju se svi 2-dimenzionalni nizovi i jedan dugi kontinuirani vektor. Taj vektor koristi se kao ulazni podatak za zadnji sloj. Potpun povezani zadnji sloj je sloj koji na temelju vrijednosti koje je dobio od prijašnjih slojeva radi predviđanje rezultata.

2.4.3. GRU

GRU (engl. *Gated Recurrent Unit*) isto je, kao i LSTM, vrsta RNN-a te u nekim slučajevima ima prednosti nad samim LSTM-om. GRU za razliku od LSTM-a koristi manje memorije i mnogo je brži, ali LSTM je mnogo precizniji kada se radi o velikim količinama podataka. GRU rješava problem koji se javlja kod RNN mreže, a to je da rješava problem nestajanja gradijenta koji je zaslužan za ažuriranje težina mreže. Do tog problema dolazi kada se mreža vraća unatrag te se težina smanjila u tolikoj mjeri da mreža ne uzima u obzir kod učenja pa samim time mrežu čini nesposobnom za učenje. GRU rješava taj problem tako da ima dvoja vrata: vrata za ažuriranje i vrata za resetiranje. Navedena vrata odlučuju koje informacije propuštaju do izlaza mreže, a koja se zadržavaju za daljnje učenje. Time se omogućava prosljeđivanje bitnijih informacija niz lanac događaja kako bi se moglo donositi bolje predviđanje.



Slika 8. *GRU model*

Slika 8. prikazuje izgled arhitekture GRU-a. Prvi dio s lijeve strane su vrata koja su zadužena za ažuriranje. U tim vratima rezultati iz ht-1 množe se sa svojom težinom i rezultati iz xt množe se sa svojom težinom te se nakon toga dodaju u sigmoidnu funkciju gdje se pretvaraju u vrijednosti između 0 i 1. Vrata ažuriranja određuju koliko će se prošlih informacija pustiti da prođu dalje kroz mrežu. Potom slijede druga vrata, a to su vrata za resetiranje. Vrata resetiranja po funkciji su slična vratima za ažuriranje, uz razliku da se ovdje ne odlučuje koje rezultate će

se propustiti dalje, već koje će se prošle informacije zaboraviti. Treći korak je korak u kojem se spajaju dva prijašnja koraka, odnosno njihovi rezultati. Rezultati se zbrajaju te se nad tim zbrojem provodi nelinearna funkcija tanh. U zadnjem koraku algoritma izračunava se vektor koji sadrži informacije i te informacije šalje dalje kroz mrežu. U ovom koraku opet sudjeluju vrata za ažuriranje koja određuju koje informacije će se pohraniti iz sadašnjega konteksta.

3. Podaci za rad

Kao što je u samom uvodu spomenuto, podaci korišteni za rad su cijene dionica američke tvrtke IBM. Podaci su prikupljani svake minute i dolaze u paketima te sadrže: vrijeme prikupljanja podatka, cijenu otvaranja dionice, cijenu zatvaranja dionice, najnižu cijenu dionice u tom razdoblju, najvišu cijenu u tom razdoblju i volumen trgovanja u tom periodu.

Tablica 1. *Prikaz prikupljenih podataka*

time	open	high	low	close	volume
2023-04-03 18:00:00	132.0	132.0	132.0	132.0	193
2023-04-03 16:00:00	132.06	132.28	132.0	132.25	121454
2023-04-03 15:00:00	132.24	132.38	131.88	132.04	801452
2023-04-03 14:00:00	132.14	132.32	132.08	132.24	232710
2023-04-03 13:00:00	132.0	132.16	131.89	132.14	220525
2023-04-03 12:00:00	132.16	132.2099	131.98	132.02	175976

Gore prikazani podaci su za dan 3. travnja 2023. godine, a način prikaza je američki te se prikaz vremena daje u satima, a ne u minutama kako će biti u podacima koji su prikupljeni za potrebe ovoga rada. Podaci su prikupljeni na način da je preuzet vlastiti API ključ koji je besplatan te promijenjene varijable kod poziva, a potom su prikupljeni potrebni podaci. Ovako izgleda poziv:

```
    CSV_URL = (
    "https://www.alphavantage.co/query?function=TIME_SERIES_INTRADAY_EXTENDED&"
    "symbol=IBM&interval=1min&slice=year1month1&apikey=IRV8L1PQARV5GPYF"
)
    with requests.Session() as s:
        download = s.get(CSV_URL)
        decoded_content = download.content.decode('utf-8')
        cr = csv.reader(decoded_content.splitlines(), delimiter=',')
```

Slika 9. *Prikaz API poziva*

Gore prikazana Slika 9. prikazuje kako se podaci zaprimaju. Prvi dio **„CSV_URL“** je URL koji se prosljeđuje kako bi se podaci mogli zaprimiti s točno određene stranice gdje su ti podaci spremljeni. U ovom slučaju radi se o podacima na dnevnoj bazi s produženom prošlošću što je vidljivo iz djela **„TIME_SERIES_INTRADAY_EXTENDED“**, nakon toga se navodi dionica koja nas zanima, u ovom slučaju **IBM**, nakon toga se odabire interval, u ovom slučaju **1 minuta** te na samom kraju **„slice“** ili odjeljak vremena iz kojega želimo da podaci stižu. Na gore prikazanoj slici **„year1month1“** što ukazuje da želimo podatke iz tekuće godine i tekućega mjeseca. Povećavanjem mjeseci i godina prikupljamo podatke iz prijašnjih mjeseci retrospektivno. Taj URL se prosljeđuje preko get metode kako bi se podaci spremili, a nakon spremanja se ti podaci moraju konvertirati s **„utf-8“** standardom. Nakon konvertiranja podatke je potrebno spremiti u **csv** format i isto tako ih podijeliti gdje se nalazi **„ , “** zbog toga što podaci dolaze u obliku **stringa** odvojeni zarezom. Na samom kraju potrebno je csv dokument pretvoriti u listu kako bi se lakše radilo s podacima, odnosno kako bi svaka minuta bio svoj element liste.

3.1. Pretprocesiranje podataka (engl. *Preprocessing*)

Kako bismo podatke mogli koristiti, potrebno ih je bilo prilagoditi unosu koji je prihvatljiv bazi i formatu koji je potreban za zadatak. Na Slici 9. zadnje dvije linije koda ukazuju nam da se u novu listu spremaju svi elementi osim prvoga, odnosno u ovom slučaju bilo je potrebno izbaciti zaglavlje jer nam nije potrebno. Uz izbacivanje zaglavlja bilo je potrebno okrenuti i samu listu jer su podaci dolazili od novijih prema starijim, a s obzirom na to da je u bazi spremljena jedna godina, nije imalo smisla da se podaci spremaju isprekidano, već moraju biti kontinuirani od godinu dana pa sve do zadnjega poziva unosa u bazu. Kada smo sve podatke pretprocesirali i prilagodili zadanoj bazi, bilo je potrebno prikupiti ih te je bilo potrebno 12 različitih API poziva kako bi se baza napunila s nešto više od 100.000 podataka. U sljedećem poglavlju bit će prikazano kako su se točno podaci spremali u bazu.

4. Baza podataka

Baza u koju su spremani podaci za potrebe ovoga rada jest *MongoDB*, točnije *Mongo Atlas* zbog toga što se radi o bazi koja se nalazi u „oblaku“ (engl. *cloud*). Kod ove vrste povezivanja s bazom bilo je potrebno imati račun na stranici **Cloud Mongo** te je nakon toga bilo potrebno pristupiti bazi, što je napravljeno na sljedeći način:

```
#Konekcija
connection_string = (
    "mongodb+srv://rafilakoseljac2001:Rafi2001@faks.lk1wuca."
    "mongodb.net/?retryWrites=true&w=majority"
)
myclient = pymongo.MongoClient(connection_string)
mydb = myclient["stocks_database"]
mycollection = mydb["stocks"]
```

Slika 10. *Povezivanje na bazu*

Na Slici 10. prikazan je u prvoj liniji *string* koji nam omogućava spajanje na bazu koja se nalazi u oblaku. Kako bi to bilo moguće, prvo je bilo potrebno instalirati samu biblioteku **„pymongo**“ koja nam omogućava spajanje te dolazi s gotovim funkcijama koje omogućavaju jednostavnije spajanje. Prilikom pozivanja funkcije **„MongoClient“** prosljeđujemo *string* koji sadrži podatke za spajanje na bazu. Nakon toga potrebno je napraviti bazu kojoj se prosljeđuje prije definirana konekcija te se još dodaje i ime baze, u ovom slučaju **„stock_database“** i na samome kraju potrebno je kreirati i kolekciju u koju će se krajnji podaci spremati, njoj se prosljeđuje baza te se dodjeljuje ime same kolekcije **„stocks“.**

4.1. Upisivanje u bazu

Prethodno je spomenuto kako će biti detaljnije objašnjeno zapisivanje u bazi. Ono izgleda ovako:

```
for row in new_list:
    if mycollection.find_one({"vrijeme": datetime.strptime(row[0],'%Y-%m-%d %H:%M:%S')}) is None:
        stocks = {
            "vrijeme": datetime.strptime(row[0],'%Y-%m-%d %H:%M:%S'),
            "open": float(row[1]),
            "high": float(row[2]),
            "low": float(row[3]),
            "close": float(row[4]),
            "volumen": int(row[5])
        }
        mycollection.insert_one(stocks)
    else:
        #Duplikat
        print(f"Unos već postoji: {datetime.strptime(row[0],'%Y-%m-%d %H:%M:%S')}")
```

Slika 11. *Spremanje u bazu*

Na Slici 11. prikazana je *for* petlja koja prolazi po ranije kreiranoj listi, nakon toga provjerava se postoji li neki unos koji ima isto vrijeme kao unos koji se upravo pokušava zapisati te ako ne postoji, nastavlja se s unosom u bazu. U slučaju da postoji, ispisuje se poruka **„Unos već postoji:“** i prikazuje se vrijeme za koje je to duplikat. Unos podatka radi se na principu da se razdvoji lista, odnosno n-torka koja sadrži podatke za svako vrijeme, tj. svaki element u listi. Najprije se unosi vrijeme koje se odmah pretvara u format **„datetime“** i sam format upisa je **„Y-m-d H:M:S“** (npr. 2023-05-01 10:01:00). Nakon toga se upisuju po redu: cijena otvaranja, najviša cijena, najniža cijena, cijena zatvaranja i volumen dionice kojom se trguje. Sve podatke osim vremena i volumena pretvara se u **float** ili decimalni broj i volumen se pretvara u **int** ili cijeli broj. Kako bi se s podacima kasnije mogla raditi predviđanja, potrebno ih je pretvoriti u neki od tih oblika jer se sa **stringom** to ne može raditi.

4.2. Dohvaćanje iz baze

Nakon spremanja podataka te je iste podatke potrebno i dohvatiti iz baze i spremiti ih u listu kako bi se s podacima mogla raditi predviđanja. To je postignuto na sljedeći način:

```
def dohvacanje_iz_baze():
    stocks = []
    for stock in mycollection.find():
        stocks.append(stock)
    stocks = pd.DataFrame(stocks)
    return stocks
```

Slika 12. *Dohvaćanje podataka iz baze*

Na Slici 12. najprije se kreira prazna lista **„stocks“** te se nakon toga pomoću *for* petlje iterira kroz prethodno kreiranu kolekciju koja je napunjena podacima te kada se nađe bilo koji podatak, on se upisuje, odnosno dodaje se u samu listu. Isti se proces ponavlja za sve podatke koji se nalaze u bazi, sve dok se svi ne spreme u listu. Nakon toga lista se pretvara u **„Pandas“** oblik, u ovom slučaju u **„DataFrame“** kako bi rad s podacima bio lakši prilikom provođenja sljedećega koraka zadatka. Funkcija nam na samom kraju vraća podatke koji su pretvoreni za rad na njima.

5. Klasifikacija – *Random Forest* model

Za strojno učenje odabrana je metoda slučajna šuma (engl. *Random Forest*) koja će klasificirati predikciju binarno 0 ako se cijena smanji i 1 ako cijena naraste. Predikcija će se provoditi na temelju cijene zatvaranja dionice. Sama funkcija predviđanja izgleda ovako:

```
def RandomForest():
    #MLFLOW
    params = {
        "n_estimators": 400,
        "min_samples_split": 20,
        "random_state": 42
    }

    stock = dohvacanje_iz_baze()
    stock = stock.drop('_id', axis=1)
    stock['vrijeme'] = stock['vrijeme'].astype(int)//10**9
    data = stock[['close']]
    data = data.rename(columns={'close': 'ac_close'})
    data["target"] = stock.rolling(2).apply(lambda x: x.iloc[1] > x.iloc[0])["open"]
    stock_pov = stock.copy()
    stock_pov = stock_pov.shift(1)
    predictors = ['close', 'open', 'high', 'low', 'volumen']
    data = data.join(stock_pov[predictors]).iloc[1:]
    model = RandomForestClassifier(**params)
    train, test = train_test_split(data, test_size=0.1, random_state=42)
    model.fit(train[predictors], train['target'])
    preds = model.predict(test[predictors])
    preds = pd.Series(preds, index=test.index)

    accuracy = accuracy_score(test['target'], preds)

    precision = precision_score(test['target'], preds)
    print("Precision: ", precision)

    rmse = mean_squared_error(test['target'], preds, squared=False)
    print("Root Mean Squared Error: ", rmse)

    print(data.head())
```

Slika 13. *Random Forest model*

Prvo je potrebno dohvatiti prethodno spremljene podatke pomoću prije navedene funkcije. Iz tih podataka izbacujemo **„_id“** koji je Mongo sam dodao prilikom upisa u bazu kako nam taj podatak ne bi smetao, isto tako vrijeme pretvaramo u cijeli broj zbog same predikcije kojoj podatak koji je u obliku dana i vremena smeta u provedbi. Podatke predikcije spremit ćemo u novo kreirani **„Data Frame“** te u njega dodati vrijeme zatvaranja kojemu mijenjamo naziv iz

„close“ u **„ac_close“**, koja nam ukazuje kolika je prava cijena koju ćemo zadržati iz izvornoga skupa. Isto tako, potrebno je kreirati cilj **„target“**, u ovom slučaju koji nam ukazuje na 0 ili 1. Nakon toga potrebno je kopirati izvorni skup i pomaknuti ga za 1 unaprijed s obzirom na to da nije moguće prikupljanje podataka iz budućnosti. Nakon toga ćemo kreirati prediktore, odnosno argumente koje želimo uzeti iz kopiranoga skupa koji je pomaknut za 1 unaprijed kako bismo ih spojili sa skupom koji smo prethodno kreirali, a koji će nam prikazivati rezultate našega strojnog učenja. Kada smo podatke spojili, potrebno je sam model pozvati. To radimo pomoću funkcije **„RandomForestClassifier“** te joj dodjeljujemo parametre: **„n_estimator“**, koja uzima 100 redova za svako predviđanje, „**min_sample_split“**, koji osigurava kako ne bismo pretrenirali naš model i **„ranodm_state“** je 1 koji uzima različite podatke kod svakoga treniranja. Kada smo model pozvali, potrebno ga je trenirati, a prije toga iskoristit ćemo **„train_test_split“** funkciju kako bismo podatke podijelili na podatke za učenje i podatke za testiranje, koje ćemo koristiti za testiranje naše preciznosti i za samopredviđanje. Nakon podjele treniramo model i kreiramo predviđanje. Kada su svi koraci napravljeni, potrebno je podatke vratiti u pandas **„Series“** oblik. Na samom kraju prikazujemo preciznost i ispisujemo prvih pet podataka našega predviđanja. Ti podaci izgledaju ovako:

```
Precision:  0.5831307275107537
Root Mean Squared Error:  0.6501493837585973
     ac_close  target       close        open        high         low  volumen
1  125.170566     1.0  123.971819  124.039135  124.067658  123.944057  11744.0
2  123.326048     0.0  125.170566  125.099258  125.180074  125.074252   2757.0
3  124.010992     1.0  123.326048  123.288016  123.496143  123.288016   4737.0
4  124.923363     1.0  124.010992  123.944057  124.039135  123.620790  22050.0
5  125.132535     1.0  124.923363  124.875824  124.961394  124.837792   9455.0
```

Slika 14. *Ispis predviđanja*

Na Slici 14. najprije je prikazana preciznost od 58,31 % koja nam ukazuje da je model malo precizniji od slučajnoga odabira. Ispod preciznosti vidljiv je ispis RMSE-a koji iznosi 0.65 te mu je prosječno odstupanje od stvarnih vrijednosti vrlo mala. Nakon toga ispisuju se predikcije, npr. 1. red **„ac_close“** ukazuje nam na broj 123.031305 $, **„target“** nam ukazuje 1, što indicira da će se cijena povisiti i nakon toga „close“ 122.650992$, što ukazuje da je predviđanje pogriješilo. Međutim, nakon toga je za 2. red predviđanje dobro pogodilo te za 3. red ponovno, ali je pogriješilo u 4. i 5. redu. Ovdje je prikazano samo prvih pet redova, ali preciznost nam ukazuje da cijeli model ima točnost od 58 %.

6. Auto regresivni integrirani pomični prosjeci

Auto regresivni integrirani pomični prosjeci slični su kao i klasifikacijski model ranije, samo što se kod ovih modela predviđala cijena dionica i koliko je njezino odstupanje od stvarne cijene. Kako bismo takve modele mogli trenirati potrebno je najprije pripremiti podatke.

```
series = dohvacanje_iz_baze().drop('_id', axis=1)
series['vrijeme'] = to_datetime(series['vrijeme'], unit='s')
series.set_index('vrijeme', inplace=True)
series = series.resample('D').ffill().fillna(method='ffill').fillna(method='bfill')

X = series['close'].values
size = int(len(X) * 0.9)
train, test = X[:size], X[size:]

history = [x for x in train]
predictions = []
```

Slika 15. *Postavljanje podataka za ARIMA i SARIMA modele*

Slika 15. prikazuje dohvaćanje podataka iz same baze te izbacivanje prvoga stupca, odnosno automatski dodan broj unosa **_id.** Nakon toga bilo je potrebno vrijeme pretvoriti u prihvatljiv oblik i postavlja se taj stupac kao indeks. Nakon toga se podaci ponovno uzrokuju (engl. *resample*) u dnevnu frekvenciju i popunjavaju se vrijednosti gdje nedostaju. Nakon toga ekstrahiramo u vrijednost **X** vrijednosti „**close**“ koja sadržava vrijednost dionice kada se ona zatvorila za taj vremenski period. Sljedeće dvije linije postavljaju veličinu za učenje i testiranje; podaci za učenje iznose 90 %, a za testiranje 10 % ukupnog skupa. Zadnje dvije linije kreiraju dvije liste **history** koja pamti podatke pomoću kojih će se model trenirati i **predictions** je prazna lista predikcija modela koje će se tu spremati. Ovaj dio koda koristit će se za ARIMA i SARIMA modele.

6.1. ARIMA Model

```
for t in range(len(test)):
    model = ARIMA(history, order=(5, 1, 0))
    model_fit = model.fit()
    output = model_fit.forecast()
    yhat = output[0]
    predictions.append(yhat)
    obs = test[t]
    history.append(obs)
    print('predicted=%f, expected=%f' % (yhat, obs))

rmse = sqrt(mean_squared_error(test, predictions))
print('Test RMSE: %.3f' % rmse)
```

Slika 16. *ARIMA model*

Slika 16. prikazuje kako teče proces treniranja i samoga rezultata ARIMA modela. Model se trenira iterativno sve dok se ne prođu svi testni podaci. Svaki poziv sastoji se od modela koji kao argumente prima podatke za učenje, odnosno **history** i parametre treniranja 5 (auto regresivni dio) označava broj kašnjenja u opažanju, 1 (integrirani dio) označava broj razlika u opažanju kako bi se vremenski niz stacionirao i 1 (dio pomičnoga prosjeka) označava veličinu prozora pomičnoga prosjeka. Nakon toga se model trenira te se predviđanja spremaju u **output** varijablu. **Yhat** varijabla sprema prvu vrijednost predviđanja te se potom sprema u prethodno kreiranu listu predikcija gdje će se kasnije spremiti sve vrijednosti predviđanja kada se prođu sve iteracije. **Obs** varijabla sprema testne podatke kako bi model u sljedećem koraku imao najnovije podatke s ciljem optimizacije modela. Svakim se korakom ispisuje stvarna vrijednost i predviđena vrijednost modela kako bi se rezultati mogli usporediti te se na samom kraju prikazuje i točnost modela koja se iskazuje pomoću RMSE (engl. *Root Mean Squared Error*) – korijena srednje kvadratne pogreške. Taj rezultat govori koliko je prosječno odstupanje predviđanja od stvarne veličine. Na Slici 17. prikazani su rezultati.

```
predicted=120.962842, expected=122.750000
predicted=122.823390, expected=122.750000
predicted=122.832011, expected=122.750000
predicted=122.730663, expected=123.250000
predicted=123.333608, expected=123.460000
predicted=123.408769, expected=125.600000
Test RMSE: 0.919
```

Slika 17. *ARIMA predikcija*

Kod ispisa rezultata na nekim je primjerima vidljivo odstupanje od stvarnoga rezultata između 0 i 2 dolara. Sam RMSE ispis govori da u prosjeku predviđanje odstupa od stvarnoga rezultata za 0,919 dolara od stvarne cijene.

6.2. SARIMA Model

```
for t in range(len(test)):
    model = SARIMAX(history, order=(5, 1, 0), seasonal_order=(1, 1, 1, 12))
    model_fit = model.fit()
    output = model_fit.forecast()
    yhat = output[0]
    predictions.append(yhat)
    obs = test[t]
    history.append(obs)
    print('predicted=%f, expected=%f' % (yhat, obs))

rmse = sqrt(mean_squared_error(test, predictions))
print('Test RMSE: %.3f' % rmse)
```

Slika 18. *SARIMA model*

SARIMA model je, kao što je i prije bilo objašnjeno, ustvari ARIMA model s dodanim elementom sezonalnosti. Na gore prikazanoj Slici 18. jedina je razlika prilikom poziva same funkcije modela pomoću SARIMAX funkcije i dodatak parametra **seasonal_order**. Prvi broj 1 označava sezonski auto regresivni poredak, drugi broj 1 označava redoslijed razlike, treći broj 1 označava poredak sezonskoga pomičnog prosjeka i broj 12 označava broj razdoblja u sezoni, u ovom slučaju 12 mjeseci u godini. Na sljedećoj slici bit će prikazani rezultati.

```
predicted=123.717989, expected=125.600000
Test RMSE: 0.906
```

Slika 19. *SARIMA predikcija*

Na Slici 19. vidljiv je samo jedan ispis stvarne i predviđene cijene zbog same razlike u ispisu kod SARIMA modela jer se prilikom treniranja ispisuju još neki parametri koji ne omogućavaju prikazivanje samo predikcija i testnih podataka. Prikazan primjer odstupa nešto manje od 2 dolara, ali sam RMSE iznosi 0,906, što model čini boljim od ARIMA modela, ali neznatno.

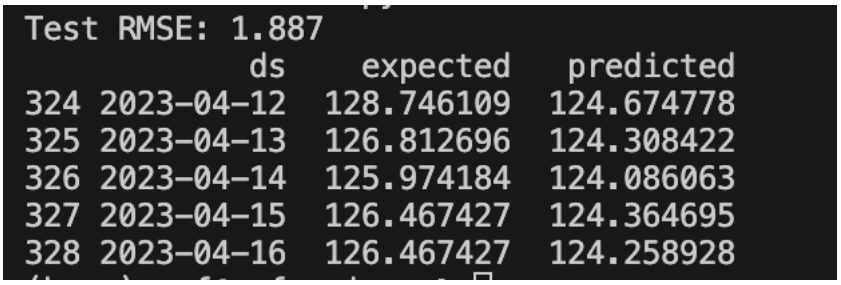
6.3. PROPHET Model

Prophet model ima vrlo sličnu pripremu podataka kao i ARIMA i SARIMA modeli, uz sitne promjene koje ne utječu na samo predviđanje modela već i na estetiku ispisa rezultata koji će se kasnije prikazati.

```
model = Prophet(
    seasonality_mode='multiplicative',
    yearly_seasonality=False,
    weekly_seasonality=True,
    daily_seasonality=True,
    changepoint_prior_scale=0.1,
    seasonality_prior_scale=10.0
)
model.fit(train)

future = model.make_future_dataframe(periods=len(test))
forecast = model.predict(future)

predictions = forecast.iloc[size:]['yhat'].values

rmse = sqrt(mean_squared_error(test['y'], predictions))
print('Test RMSE: %.3f' % rmse)

comparison = pd.DataFrame({'ds': test['ds'], 'expected': test['y'], 'predicted': predictions})
print(comparison.head())
```

Slika 20. *Prophet model*

Slika 20. prikazuje proces treniranja Prophet modela. Pozivom funkcije Prophet moramo navesti parametre koji će se koristiti za učenje. Prvi od njih je **seasonality_mode** koji je postavljen na **multiplicative**, što označava da se sezonalnosti množe umjesto da se zbrajaju ili da koriste neku drugu od mogućih opcija. Sljedeće tri linije označavaju koja sezonalnost će se koristiti, u ovome su slučaju to tjedna i dnevna sezonalnost. Nakon toga **checkpoint_prior_scale** označava koliko je model fleksibilan na trendove te što je manji broj, stroža je provjera i smanjuje šansu od pretreniranja modela. Zadnji parametar **seasonality_prior_scale** označava koliko je model fleksibilan na sezonalne komponente, u ovom slučaju broj 10 označava veću varijabilnost modela na sezonalne podatke. Nakon toga model se trenira s podacima za učenje te se kreira novi skup podataka proširen podacima za testiranje te se taj skup proširuje sa stupcem gdje su iskazana vremena za predviđanja. Kada se kreirao budući skup podataka, provodi se predviđanje nad njime te se potom rezultati koji su se kreirali sijeku tako da odgovaraju vremenima u kojima se nalaze testni podaci. Tim postupkom moguće je vidjeti koliko je model precizan ili neprecizan tako da se uspoređuju stvarni podaci.



Slika 21. *Prophet predikcija*

Slika 21. prikazuje nam to predviđanje. RMSE iznosi 1,887, što označava da je prosječno odstupanje nešto manje od 2 dolara. Na primjerima ispod RMSE rezultata rezultati variraju između 1 pa sve do 4 dolara.

7. Neuronska mreža

Budući da je rezultat koji daje prijašnji model u obliku 0 i 1, odnosno 0 kad cijena dionice pada i 1 kad cijena dionice raste, težilo se izradi modela koji može predvidjeti rezultat u obliku grafa kao što se uobičajeno i rade predviđanja nad cijenama dionica. Kako bi se to postiglo, kreiran je model neuronskih mreža. Naravno, podatke koji su korišteni za prijašnji model bilo je potrebno procesirati na drukčiji način kako bi neuronske mreže mogle te podatke koristiti. Prvi dio je isti kao i kod prijašnjega modela, gdje podatke preuzimamo s MongoDB-a te ih nakon toga dijelimo na podatke za učenje i testiranje. Nakon toga bilo ih je potrebno normalizirati, što je odrađeno pomoću funkcije **MinMaxScaler()** koja brojeve pretvara u interval od 0 do 1. Normalizirane podatke razdvajamo na X i y i nakon toga ih preoblikujemo tako da se mogu koristiti kao ulaz u model.

```
stock = dohvacanje_iz_baze()
stock = stock.drop('_id',axis=1)
stock['vrijeme']=stock['vrijeme'].astype(int)//10**9
stock_pov = stock.copy()
stock_pov = stock_pov.shift(1)
predictors = ['close','open','high','low','volumen']
data = stock_pov[predictors].iloc[1:]
data_test = stock_pov[predictors].iloc[1:]
data_test = data_test.iloc[90000:,]
#print(data.head())
training_set = data.iloc[:,1:2].values
test_set = data_test.iloc[:,1:2].values
#print(training_set)
#print(training_set.shape)
scaler = MinMaxScaler(feature_range=(0,1))
scaled_training_set = scaler.fit_transform(training_set)
#print(scaled_training_set)
X_train = []
y_train = []
for i in range(60,1258):
    X_train.append(scaled_training_set[i-60:i, 0])
    y_train.append(scaled_training_set[i,0])
X_train = np.array(X_train)
y_train = np.array(y_train)
#print(X_train.shape)
#print(y_train.shape)
X_train = np.reshape(X_train,(X_train.shape[0],X_train.shape[1],1))
```

Slika 22. *Dohvaćanje podataka iz baze za neuronske mreže*

Slika 22. prikazuje nam kako je teklo procesiranje podataka. Nakon što smo podatke procesuirali, bilo je potrebno kreirati sam model i istrenirati ga. Sljedeća slika prikazuje taj proces.

7.1. LSTM model

```
modelML = Sequential()
modelML.add(LSTM(units=50, return_sequences=True, input_shape = (X_train.shape[1],1)))
modelML.add(Dropout(0.2))
modelML.add(LSTM(units=50,return_sequences=True))
modelML.add(Dropout(0.2))
modelML.add(LSTM(units=50,return_sequences=True))
modelML.add(Dropout(0.2))
modelML.add(LSTM(units=50))
modelML.add(Dropout(0.2))
modelML.add(Dense(units=1))
modelML.compile(optimizer='adam',loss = 'mean_squared_error')
modelML.fit(X_train,y_train,epochs=100,batch_size=32)
```

Slika 23. *LSTM model*

LSTM model je model s devet slojeva, a radi se o tri različita sloja, to su **LSTM sloj, Dropout sloj i Dense sloj**. **LSTM sloj** služi kako bi se smanjilo vrijeme treniranja jer taj sloj bira koji su resursi dostupni u to vrijeme i bira one koji zadatak odrađuju najbrže. **Dropout sloj** služi kako ne bismo naš model pretrenirali pa određeni postotak podataka otpada prilikom njegova izvođenja. Zadnji sloj je **Dense sloj** ili potpuno povezani sloj koji služi kako bi primio podatke otprije i kako bi ih sve povezao u jedan izlaz. Na kraju koristimo optimizacijsku funkciju **adam** i funkciju gubitka **mean_squared_error**. Učenje će trajati 100 epoha s 32 uzorka u seriji (engl. *batch size*) tijekom prolaza kroz mrežu.

```
dataset_total = pd.concat((data['open'],data_test['open']),axis=0)
inputs = dataset_total[len(dataset_total)-len(data_test)-60:].values
inputs = inputs.reshape(-1,1)
inputs = scaler.transform(inputs)
X_test = []
for i in range(600,10000):
    X_test.append(inputs[i-60:i,0])
X_test = np.array(X_test)
X_test = np.reshape(X_test,(X_test.shape[0],X_test.shape[1],1))
pred = modelML.predict(X_test)
pred = scaler.inverse_transform(pred)
```

Slika 24. *Kreiranje predviđanja*

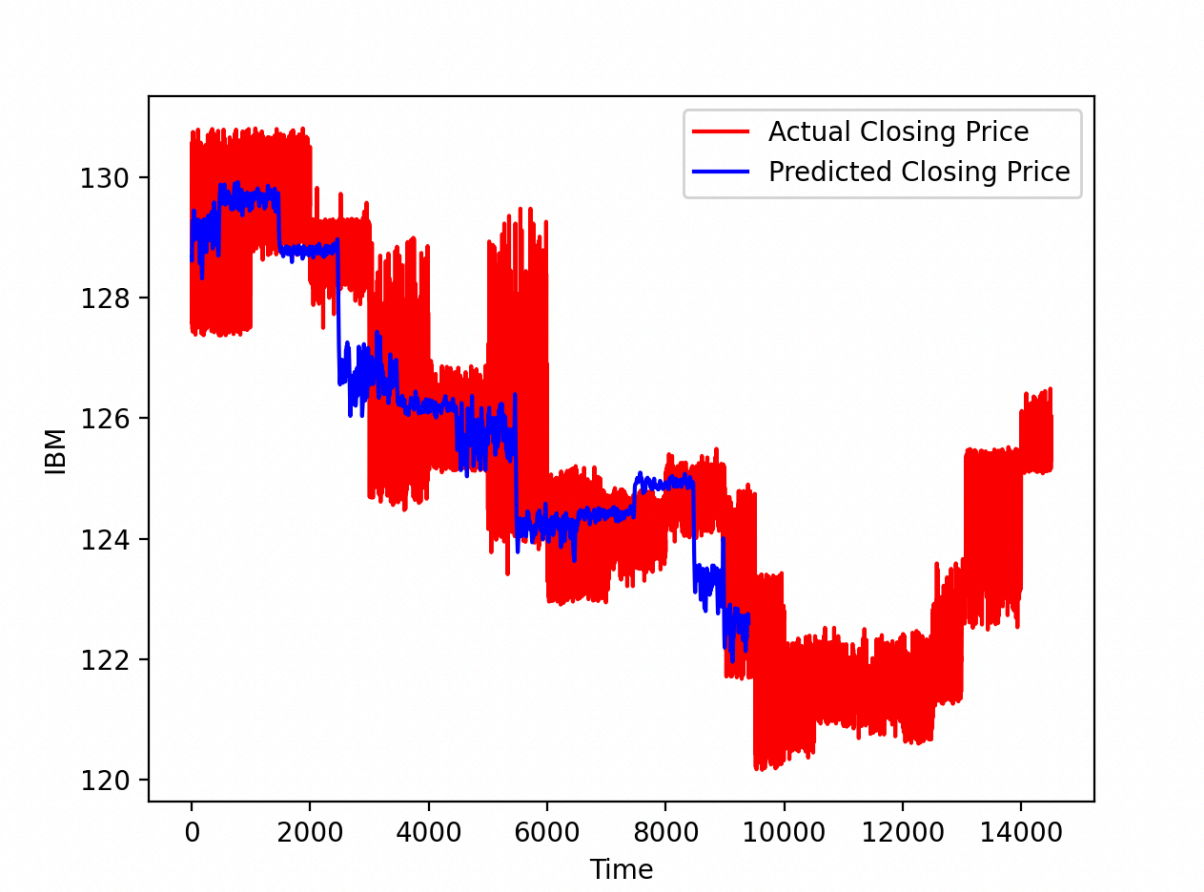
Slika 24. prikazuje sljedeći korak, a to je predviđanje. Kako bismo izradili predviđanje, potrebno je spojiti podatke za učenje i testiranje da bismo dobili broj ulaza, odnosno da bismo ukupan broj mogli oduzeti od testnih podataka i 60 koje će se koristiti za predviđanje. Nakon toga je podatke potrebno normalizirati kako bi se mogli, kao i ranije, koristiti kao ulaz u model, ali se sada koriste za kreiranje predikcije. Nakon toga podatke za testiranje provlačimo kroz

funkciju **predict()** te nad predikcijom provodimo **inverse_transform()** funkciju koja generira nasumične brojeve koje koristi kako bi izračunao distribuciju podataka. Kako bismo vidjeli rješenje, potrebno je kreirati graf koji će služiti kao usporedba stvarnih podataka i predviđanja koje je model sam napravio.

```
plt.plot(test_set,color = 'red',label = 'Stvarna cijena dionica')
plt.plot(pred,color = 'blue',label = 'Predvidena cijena dionica')
plt.xlabel('Time')
plt.ylabel('IBM')
plt.legend()
plt.show()
```

Slika 25. *Kreiranje dijagrama*

Slika 25. prikazuje kod koji kreira dva grafa – jedan crvene boje koji koristi testne podatke i drugi plave boje koje je model sam kreirao kao predikciju. Sljedeća slika nam to prikazuje:



Slika 26. *LSTM graf*

Iz Slike 26. vidljivo je da naša predikcija, odnosno plavi graf unaprijed predviđa kako će graf izgledati. Naravno, ima nekih odstupanja, ali iz grafa se može očitati u kojem smjeru se kreće cijena prije nego što do te cijene dođe.

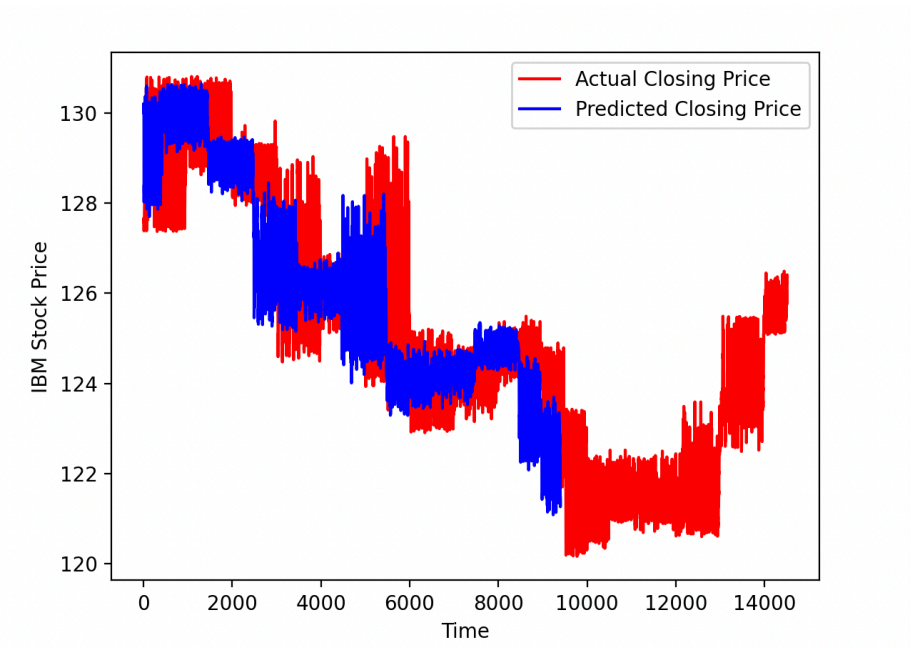
7.2. CNN model

Za drugi je model odabran je **CNN** ili **model konvolucijske neuronske mreže**. Što se tiče same izvedbe modela za pripremu i dohvaćanje podataka, korištene su iste metode kao i kod LSTM modela. Jedina je razlika model.

```
model_cnn = Sequential()
model_cnn.add(Conv1D(filters=64, kernel_size=3, activation='relu', input_shape=(X_train.shape[1], 1)))
model_cnn.add(MaxPooling1D(pool_size=2))
model_cnn.add(Flatten())
model_cnn.add(Dense(units=50, activation='relu'))
model_cnn.add(Dense(units=1))
```

Slika 27. *CNN model*

U modelu su korišteni sljedeći slojevi: **Conv1D, MaxPooling1D, Flatten** i 2 **Dense** sloja. **Conv1D** je konvolucijski sloj koji se koristi kod vektora ili nizova, kao u ovom slučaju. Ovaj sloj radi na principu množenja i zbrajanja s drugim nizovima ili vektorima kako bi dobio rezultat. **MaxPooling1D** je sljedeći sloj koji uzima matricu veličine, u ovom slučaju 2x2 i iz te matrice uzima najveći rezultat, odnosno najbolji. **Flatten** sloj služi kako bismo smanjili dimenzije naše mreže. I na kraju imamo 2 **Dense** sloja koji su potpuno povezani slojevi. Rezultat je sljedeći:



Slika 28. *CNN graf*

Na grafu (Slika 28.) vidljivo je kako predviđanje modela vjernije i bliže prati same cijene dionica, ali opet kao i kod LSTM-a vrhovi i dolovi nisu iste visine. Sada ćemo prikazati posljednji model pa usporediti rezultate.

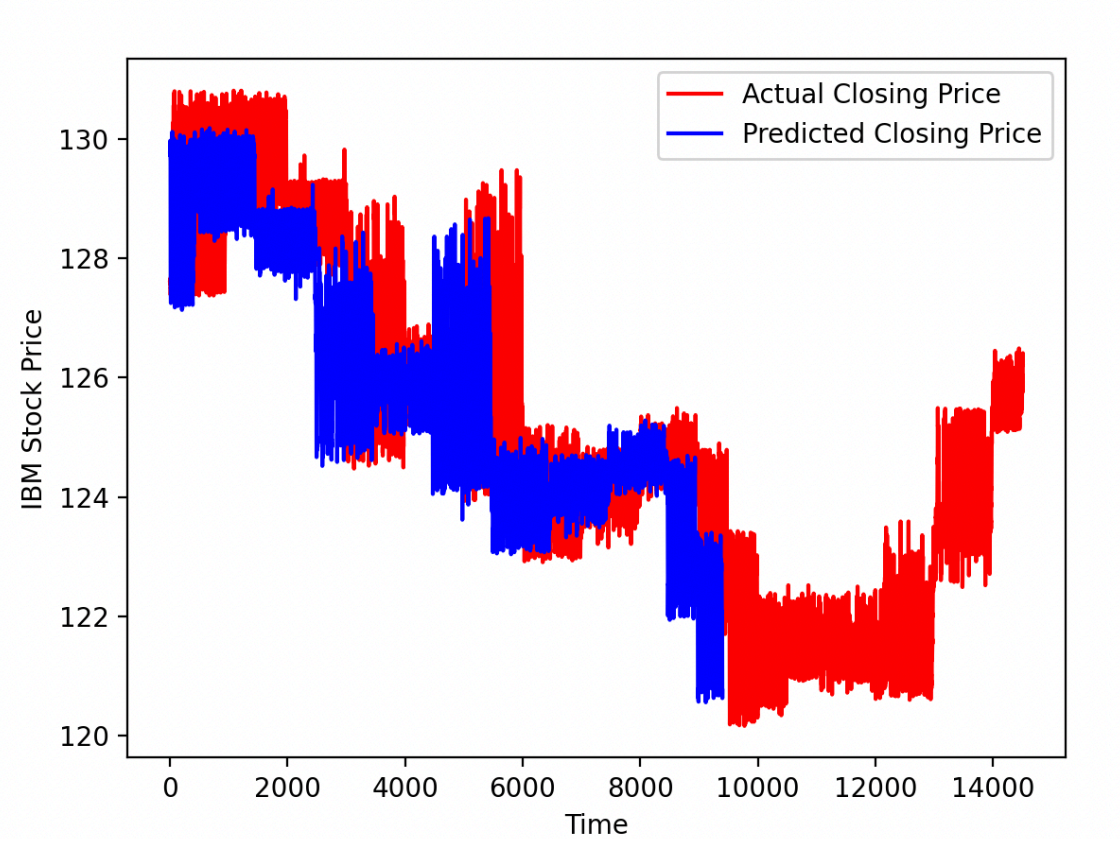
7.3. GRU model

Kao posljednji model odabran je **GRU model** koji je sličan LSTM modelu jer koristi **long short-term memory**, ali koristi manje parametara i mnogo je fleksibilniji, no nedostatak mu je da učenje traje dugo i lako je moguće pretrenirati sam model. Kao i kod prijašnjih modela sve su zadane postavke iste.

```
model_gru = Sequential()
model_gru.add(GRU(units=50, return_sequences=True, input_shape=(X_train.shape[1], 1)))
model_gru.add(Dropout(0.2))
model_gru.add(GRU(units=50, return_sequences=True))
model_gru.add(Dropout(0.2))
model_gru.add(GRU(units=50, return_sequences=True))
model_gru.add(Dropout(0.2))
model_gru.add(GRU(units=50))
model_gru.add(Dropout(0.2))
model_gru.add(Dense(units=1))
```

Slika 29. *GRU model*

GRU model koristi tri različita sloja, a to su **GRU, Dropout** i **Dense**. **GRU** sloj je rekurentni sloj koji koristi u nizovima, kao što je cijena dionica jedan od vremenskih nizova. Ovaj sloj uči o odnosima između podataka. **Dropout** sloj koristi se kao i kod LSTM modela kako bi se smanjilo pretreniranje. Na kraju, kao i kod svih modela, dolazi **Dense** sloj kao potpuno povezani kako bi se dobio rezultat.



Slika 30. *GRU graf*

Kao što je i vidljivo, GRU model daje najbolje podatke i oni su najbliži stvarnim cijenama dionica. Ovaj je model vrlo sličan LSTM modelu i po parametrima i po slojevima, no daje mnogo bolje rezultate od njega, a mnogo bolje rezultate daje CNN uz manje slojeva. Za sam kraj prikazat će se još i unaprijeđeni model GRU.

7.4. GRU model (poboljšani)

Budući da je GRU model najbolje predvidio cijenu dionica, pokušat ćemo malo promijeniti parametre kako bismo dobili još bolje rezultate.

```
X_train = []
y_train = []
for i in range(120, len(scaled_training_set)):
    X_train.append(scaled_training_set[i - 120:i, 0])
    y_train.append(scaled_training_set[i, 0])
X_train, y_train = np.array(X_train), np.array(y_train)
```

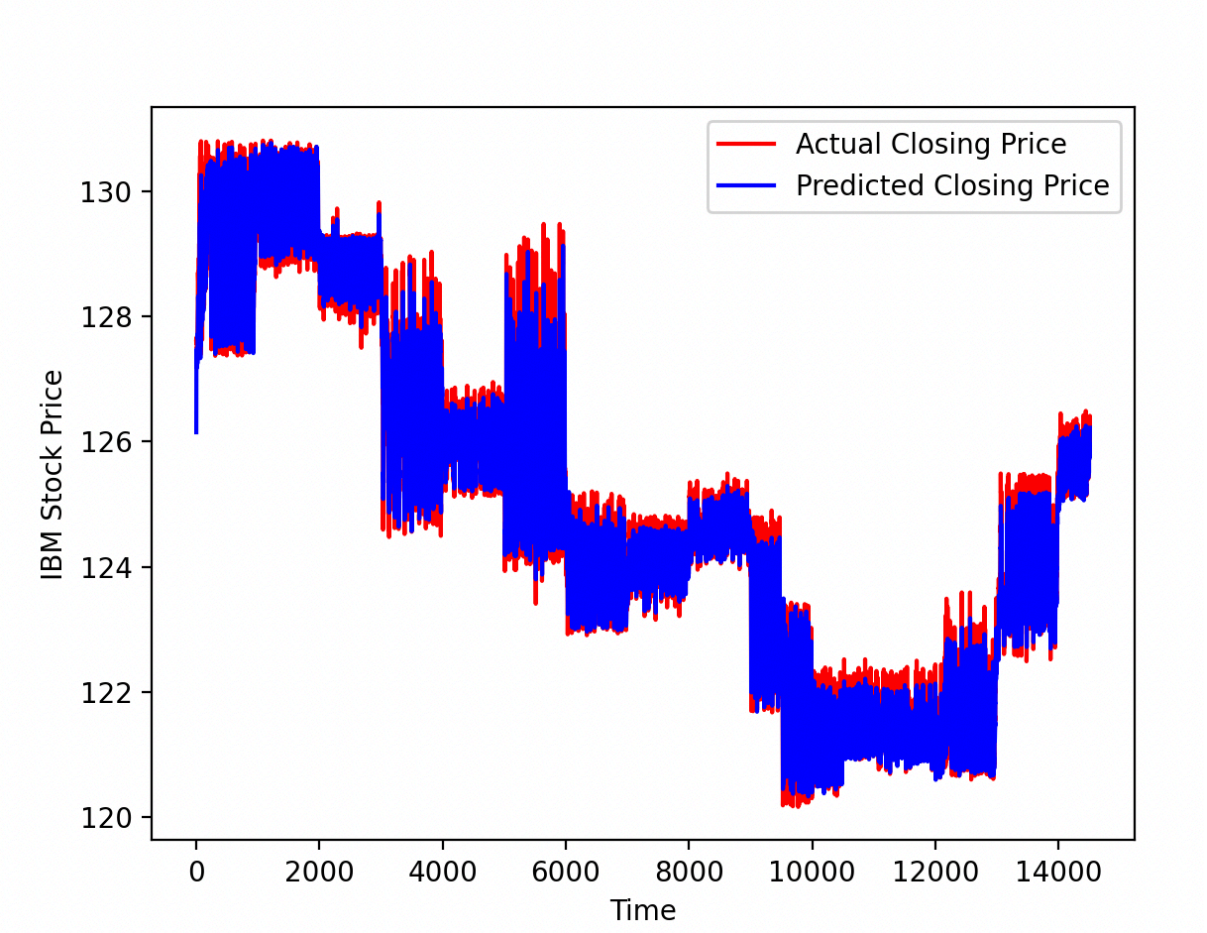
Slika 31. *Promijenjeni parametri za GRU model*

Na Slici 31. vidljiva je promjena dužine s prijašnjih 60 na 120.

```
model_gru = Sequential()
model_gru.add(GRU(units=100, return_sequences=True, input_shape=(X_train.shape[1], 1)))
model_gru.add(Dropout(0.2))
model_gru.add(GRU(units=100, return_sequences=True))
model_gru.add(Dropout(0.2))
model_gru.add(GRU(units=100))
model_gru.add(Dropout(0.2))
model_gru.add(Dense(units=1))
```

Slika 32. *GRU model 2*

Na Slici 32. nalaze se parametri za poboljšani model. S prijašnjih 50 slojeva podignulo se na 100 slojeva na svakom GRU sloju. Što se tiče samog kreiranja modela, broj epoha smanjio se zbog vremena izvođenja jer je prijašnjem modelu trebala 1 do 2 sekunde, a sada s novim modelom 4 do 5 minuta po epohi te se zbog toga novi model vrti u 10 epoha umjesto 100, što je bio slučaj u prijašnjem modelu.



Slika 33. *Dotrenirana verzija GRU grafa*

Na Slici 33. vidljivo je znatno poboljšanje predviđanja gdje se rezultati skoro pa 100 % poklapaju sa stvarnim cijenama dionica. Jedina razlika od prijašnjih modela je da se kod ovoga modela cijena nalazi u istom vremenskom intervalu kako se i cijena stvarno ponaša, a kod

prijašnjih modela predviđena je cijena bila lijevo, odnosno prije nego što se cijena na tržištu formirala.

8. Usporedba rezultata

U ovoj sekciji pobliže će se usporediti dobiveni rezultati svih modela.

Klasifikacijski model slučajne šume dao je preciznost od **58 %**, što ga ne čini vrlo preciznim. Uz to dobiveni RMSE rezultat od **0,65** ukazuje da je odstupanje predviđanja od stvarnih vrijednosti nisko, ali problem je u tome što se dobiveni rezultati odnose na vremenski interval od 1 minute, a samo učenje modela traje nešto duže. Da bi sam model bio koristan, trebalo bi kreirati graf s rezultatima ili usporediti rezultate s modelom koji predviđa neki drugi vremenski interval kako bi se mogle donositi odluke vezane uz cijene dionica.

ARIMA i SARIMA modeli, slično kao i kod klasifikacijskoga modela, daju rezultate za minutni vremenski interval, što ga ne čini idealnim rješenjem. Rezultati ARIMA i SARIMA modela ukazuju na RMSE **0,919** kod ARIMA modela i **0,906** kod SARIMA modela. Rezultati u usporedbi s klasifikacijskim modelom govore da ti modeli imaju veće odstupanje od klasifikacijskoga modela, ali je samo učenje modela upola brže.

Prophet model ima najgore rezultate od svih prijašnjih te mu RMSE iznosi **1,887**, što ga čini lošijim odabirom od prijašnjih modela, a brzinom je podjednak ARIMA i SARIMA modelima. Neuronske mreže prikazuju nam grafički rezultat što kod trgovanja dionicama uvelike pomaže jer su i sami podaci o dionicama prikazani u obliku grafa.

Prvi od tri modela je LSTM koji je od tri vrste neuronskih mreža imao najgori rezultat. Brjegovi i dolovi na grafu nisu dostizali visine koje su imali stvarni podaci, ali je predvidio smjer kretnja dionica, što ga ne čini bezvrijednim. Samo učenje traje nekoliko minuta pa ga to ne čini idealnim, ali vidljiv je graf koji predviđa tisuće minuta unaprijed.

Drugi po redu je CNN model koji je rezultatom bio vrlo sličan LSTM modelu, ali su njegovi brjegovi i dolovi nešto bliže stvarnim vrijednostima.

Zadnji model neuronskih mreža je GRU model koji je sa sličnim parametrima kao prijašnji modeli neuronskih mreža dobio najbolje rezultate. GRU model najbliže je predvidio kretanje cijena dionica u istom vremenu treniranja kao prijašnji modeli neuronskih mreža.

Nakon razmatranja rezultata svih modela kreiran je posljednji model koji je bio dotrenirana verzija GRU modela. Toj verziji modela postavljeni su parametri tako da učenje traje duže kako bi se postigao što bolji rezultat. Učenje traje 20-ak minuta, ali rezultati na prošloj slici (Slika 33.) prikazuju da je model uz sitne razlike sa stvarnim podacima uspio predvidjeti cijenu dionica za interval od 14.000 minuta. Uz to, rezultati ovoga modela pozicionirali su graf

direktno iznad stvarne cijene dionica, što prijašnji modeli nisu uspjeli, već su bili pozicionirani s lijeve strane podataka. Takav model mogao bi se dalje optimizirati. Odnosno s dodavanjem podataka i dužim vremenom treniranja bio bi još precizniji.

9. Verzioniranje – MLflow

U ovoj sekciji prikazat će se kako alat MLflow spaja s Visual Studio Codeom te kako je sam integriran u okruženju. Kako bi se MLflow koristio potrebno ga je instalirati pomoću naredbe **pip install mlflow.** Nakon toga potrebno je biblioteku navesti u kodu kako bi se mogla koristiti. Potrebno je odvojiti parametre našega modela izvan samoga poziva modela kako bi se ti modeli mogli spremiti pomoću MLflowa.

```
params = {
    "n_estimators": 400,
    "min_samples_split": 20,
    "random_state": 42
}
```

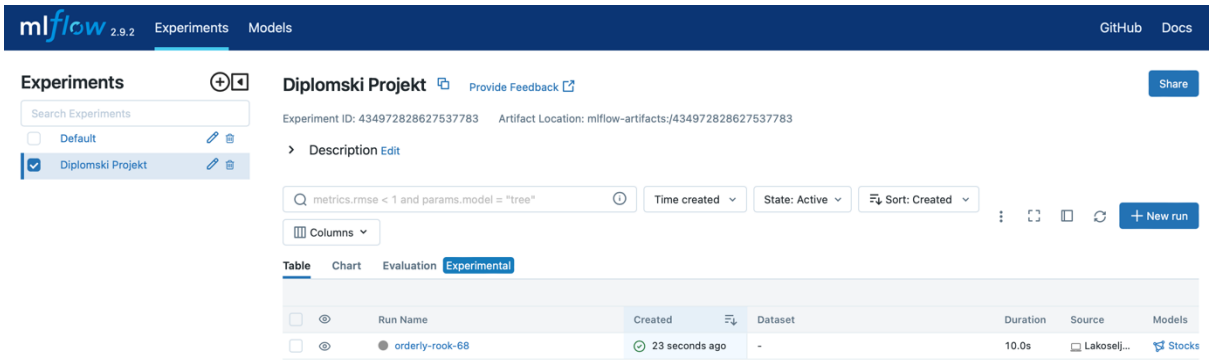
Slika 34. *Parametri za verzioniranje*

Na Slici 34. prikazani su parametri koje model koristi, a koje smo prethodno prikazali, samo što su sada odvojeni kako bi se mogli spremiti. Da bi MLflow radio, potrebno je pokrenuti server u terminalu naredbom **mlflow server --host 127.0.0.1 --port 8080**. Naredba može varirati ovisno o portu i IP adresi.

```
mlflow.set_tracking_uri(uri="http://127.0.0.1:8080")
mlflow.set_experiment("Diplomski Projekt")
with mlflow.start_run():
    # Log the hyperparameters
    mlflow.log_params(params)

    # Log the loss metric
    mlflow.log_metric("accuracy", accuracy)
    mlflow.log_metric("presicion", precision)
    mlflow.log_metric("root mean squared error", rmse)

    # Set a tag that we can use to remind ourselves what this run was for
    mlflow.set_tag("Training Info", "Stock prices Random forest")
    # Infer the model signature
    signature = infer_signature(train[predictors], model.predict(train[predictors]))

    model_info = mlflow.sklearn.log_model(
    sk_model=model,
    artifact_path="Stocks_model",
    signature=signature,
    input_example=train[predictors],
    registered_model_name="Stocks",
    )
    loaded_model = mlflow.pyfunc.load_model(model_info.model_uri)

predictions = loaded_model.predict(test[predictors])

result = pd.DataFrame(test[predictors])
result["actual_class"] = test['target']
result["predicted_class"] = predictions

result[:4]
```

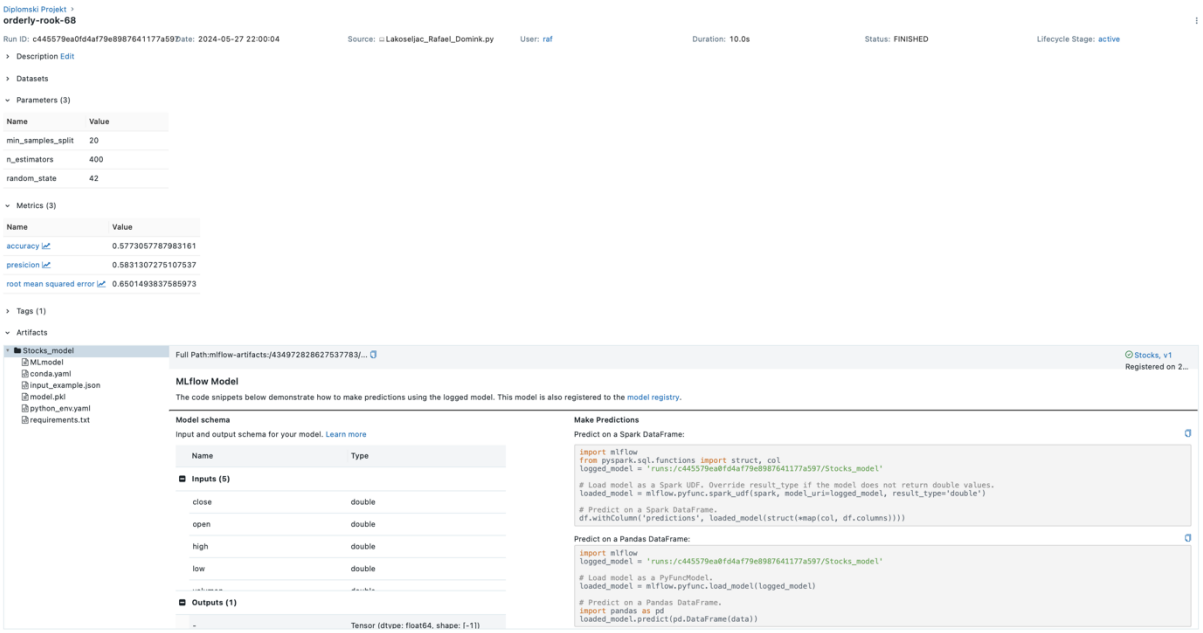
Slika 35. *Kreiranje MLflow poveznice*

Slika 35. prikazuje kreiranje adrese na koju će se sve informacije za MLflow spremiti i kojoj se može pristupiti tim podacima. Nakon toga dajemo ime našem eksperimentu i potom pokrećemo sam MLflow. Naredbom **mlflow.start_run()** pokrećemo funkciju koja otvara mogućnost zapisa i spremanja informacija o modelu te sam model. Funkcija **log_params(params)** služi za spremanje parametara za naš model, funkcija **log_metrics("accuracy",accuracy)** sprema rezultat preciznosti koji naš model postiže i funkcija **set_tag()** služi kako bismo nazvali sam *tag* eksperimenta. Varijabla **signiture** služi nam kako bismo spremili ulaze, izlaze i parametre modela koje ćemo kasnije spremiti. Varijabla **model_info** služi za pozivanje funkcije **log_model()** pomoću koje spremamo model, artefakte, potpis, odnosno signiture koji smo prije naveli, ulaze i samo ime modela. Nakon toga sam model učitavamo i koristimo za predikciju i za ispisivanje rezultata. Sada ćemo prikazati kako to izgleda u samom MLflowu. Nakon što smo pokrenuli prethodno spomenutu naredbu u terminalu, dobivamo sljedeći prozor na adresi **http://localhost:8080:**



Slika 36. *Prikaz eksperimenata na MLflow sučelju*

Slika 36. prikazuje sučelje koje nas dočekuje kada prvi put otvorimo *localhost* adresu. S lijeve strane je padajući izbornih svih naših eksperimenata i kada odaberemo eksperiment koji nas zanima, na sredini su prikazane sve verzije našega modela, u ovome slučaju imamo dvije verzije. Otvaranjem jedne od tih verzija dobivamo sljedeći prozor:



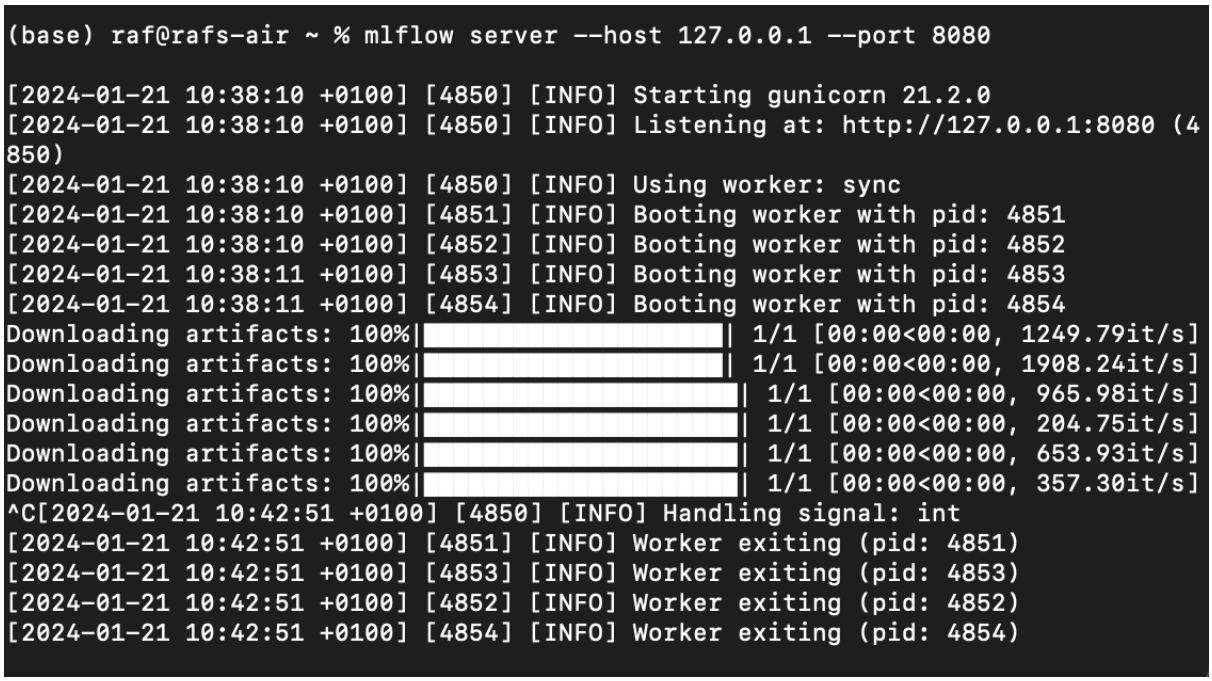
Slika 37. *Prikaz određene verzije modela*

Na Slici 37. možemo vidjeti sve informacije koje smo prije spremili. Vidljivi su parametri, metrike i tagovi koje smo naveli prije. Ispod toga vidljiv je izbornik koji prikazuje sam model, yaml datoteku koja prikazuje sve što nam treba od biblioteka kako bismo pokrenuli sam model, json datoteku koja nam sprema prije spomenuti izlaz, odnosno predviđanje, model.pkl omogućava nam skidanje samoga modela te ispod toga Python verziju koju moramo imati kako bi model mogao raditi. Prilikom pokretanja programa u *Visual Studio Codeu* dobivamo sljedeći prozor koji potvrđuje slanje modela pomoću MLflowa:

```
Registered model 'Stocks' already exists. Creating a new version of this model...
2024/01/21 10:59:20 INFO mlflow.store.model_registry.abstract_store: Waiting up to 300 secon
ds for model version to finish creation. Model name: Stocks, version 2
Created version '2' of model 'Stocks'.
Downloading artifacts: 100%|                                        | 6/6 [00:00<00:00, 82.96it/s]
(base) raf@rafs-air APVO_projekt %
```

Slika 38. *Konzolni prozor kreiranja modela*

Slika 38. prikazuje koja je verzija modela i njegovo ime te ako model ne postoji, on bi se tu prikazao i automatski kreirao. U ovom je slučaju to 2. verzija pa nije potrebno ponovno kreiranje.



Slika 39. *Ispis servera za MLflow*

Slika iznad prikazuje ispis u samome terminalu koji prikazuje prijenos i skidanje artefakata na MLflowu.

Zaključak

Ovim je radom pristupljeno vrlo zanimljivome problemu koji na samom kraju daje rezultate koji se mogu primijeniti pri trgovanju dionicama. Isto tako, može se primijeniti na trgovanje valutama i slično. Iz samoga je rada vidljivo da je bilo potrebno vrlo malo kako bi se podaci dohvatili. Naravno, najviše vremena oduzelo je pronalaženje kvalitetnog API-ja koji sadrži podatke koji su primjenjivi i s kojima se lako može baratati. Nakon toga jedini problem s kojim smo se susreli bilo je sporo upisivanje velike količine podataka u bazu zbog same usluge koju pruža MongoDB Atlas. Razlog tomu je rad na besplatnoj verziji, ali taj problem riješio bi se tako da se uzme dediciran server koji bi baratao s našim podacima te bi, uz navedeni server, zapisivanje i čitanje podataka bilo puno brže i efikasnije.

Valja spomenuti da nas ovaj rad na zanimljiv način uči mnogo o tome kako se sami podaci mogu prikupiti i pruža novu vrstu baza podataka. Korištene su baze puno fleksibilnije što se tiče vrsta unosa te se na tim bazama može raditi mnogo primjena u različitim djelatnostima. Modeli neuronskih mreža radili su mnogo bolje i samo prikazivanje njihovih rezultata korisnije je kada je vidljivo na grafu s obzirom na to da se same cijene dionica tako i prikazuju.

GRU model prikazan je kao najbolji model koji će dati najbolje rezultate s jednostavnim parametrima, a kasnije s podešenim parametrima rezultat je bio još bolji. Jedini problem kod boljega modela jest vrijeme potrebno za učenje. Taj problem mogao bi se riješiti da se model trenira jednom tjedno ili jednom dnevno te da ga se spremi i koristi. To rješenje smanjilo bi vrijeme koje bi bilo potrebno da se dobije rezultat predviđanja s obzirom na to da je vrijeme vrlo bitan faktor u svijetu dionica.

Literatura

Aplha Vantage API Documentation, pristupljeno(3.5.2024.), Dostupno na: https://www.alphavantage.co/documentation/

Classification in Machine Learning: An Introduction (pristupljeno 9. 5. 2024.). Dostupno na: https://www.datacamp.com/blog/classification-machine-learning

Convolutional Neural Network Tutorial (pristupljeno 15. 5. 2024.). Dostupno na: https://www.simplilearn.com/tutorials/deep-learning-tutorial/convolutional-neural-network

Forecasting at scale. (pristupljeno 14. 5. 2024.). Dostupno na: https://facebook.github.io/prophet/

Lakoseljac, Rafael Dominik (2024.) *Predviđanje cijene dionica*. Seminarski rad u sklopu kolegija *Meko računarstvo*, pod mentorstvom izv. prof. dr. sc. M. Ivašić-Kos te asistenata K. Host i A. Poleksić

MongoDB Documentation (pristupljeno 2. 5. 2024.). Dostupno na: https://www.mongodb.com/docs/

Portofolio Project: *Predicting Stock Prices Using Pandas and Scikit-learn* (pristupljeno 3.5.2024.). Dostupno na: https://www.dataquest.io/blog/portfolio-project-predicting-stock-prices-using-pandas-and-scikit-learn/

Stock Price Prediction Using Machine Learning (pristupljeno 3. 5. 2024.). Dostupno na: https://www.simplilearn.com/tutorials/machine-learning-tutorial/stock-price-prediction-using-machine-learning

Time Series Analysis: Definition, Types, Techniques, and When It's Used (pristupljeno 5. 5. 2024.). Dostupno na: https://www.tableau.com/learn/articles/time-series-analysis

Time Series Forecasting: Definition, Applications, and Examples (pristupljeno 8. 5. 2024.). Dostupno na: https://www.tableau.com/learn/articles/time-series-forecasting

Understanding LSTM Networks (pristupljeno 15. 5. 2024.). Dostupno na: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

What Is ARIMA Modeling? (pristupljeno 12. 5. 2024.). Dostupno na: https://www.mastersindatascience.org/learning/statistics-data-science/what-is-arima-modeling/

What is Segmentation in time- series or statistical analysis? (pristupljeno 5. 5. 2024.). Dostupno na: https://questdb.io/glossary/segmentation/

What is time series classification? (pristupljeno 5. 5. 2024.). Dostupno na:

https://developer.ibm.com/learningpaths/get-started-time-series-classification-api/what-is-time-series-classification/

Popis slika

Popis tablica