VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS
DUOMENŲ MOKSLO STUDIJŲ PROGRAMA

Bakalaurinis darbas

Laiko eilučių modelių modeliavimas panaudojant natūralios kalbos modelius

Learning Transferable Time-Series Models From Natural Language Supervision

Darbo atliko: Matas Gaulia
Darbo vadovas: doc. dr. Linas Petkevičius
Darbo recenzentė: doc. dr. Jurgita Markevičiūtė

VILNIUS 2023

Turinys

Darbo autorius dėkoja Vilniaus universiteto Matematikos ir informatikos fakulteto Informacinių technologijų atviros prieigos centrui už suteiktus HPC išteklius šio darbo tyrimams atlikti.

Lentelių sąrašas

Iliustracijų sąrašas

Laiko eilučių modelių modeliavimas panaudojant natūralios kalbos modelius

Santrauka

Šis darbas tyrinėja ryšį tarp kriptovaliutų laiko eilučių duomenų ir Twitter žinučių apie jas. Konvertuojant laiko serijas į paveikslėlius, originalus CLIP modelis yra apmokomas suvokti naujus ryšius. Taip pat buvo pasiūlytas modifikuotas CLIP modelis, kuris priima laiko serijas tiesiogiai, be jokio konvertavimo.

Raktiniai žodžiai : Kriptovaliutos; Laiko eilutės; Natūralios kalbos apdorojimas

Learning Transferable Time-Series Models From Natural Language Supervision

Abstract

This thesis researches relationship between cryptocurrency time series data and Twitter's tweets about them. By converting time series into images, the original CLIP model is trained to understand new relationships. Then a modified CLIP model is proposed that accepts time series directly without the need for any conversion.

Key words : Cryptocurrencies; Time series; Natural language processing

Acronyms

BTC Bitcoin kriptovaliuta. 1

CLIP Contrastive Language-Image Pre-Training. 1

ETH Ether kriptovaliuta. 1

XRP Ripple kriptovaliuta. 1

1 Įvadas

Šiame bakalauro darbe nagrinėjamas naujas požiūris į laiko eilučių analizę, naudojant kriptovaliutų duomenis ir „Twitter“ žinučių duomenis. Šiame tyrime siekiama išsiaiškinti, ar šiuolaikiniai dirbtinio intelekto modeliai, tokie kaip CLIP [RKH+21], gali būti pritaikyti laiko eilučių prognozavimui ir analizei, naudojant natūralią kalbą kaip pagrindinį supervizijos šaltinį.

Laiko eilučių analizė yra svarbus finansų [SGO20], meteorologijos [Cra65], medicinos [Ayd22], energetikos [ZPHW13] ir kitų sričių tyrimo metodas, leidžiantis prognozuoti būsimą elgseną, tendencijas ir reiškinius. Kriptovaliutų rinka yra ypač sudėtinga dėl savo nestabilumo, greito augimo ir pokyčių. Todėl efektyvus laiko eilučių modeliavimas šioje srityje gali padėti investuotojams, analitikams ir finansų ekspertams priimti geriausius sprendimus.

Šiame darbe naudojamas CLIP modelis, kuris yra vienas iš pažangiausių giliojo mokymosi modelių, pasiūlytas „OpenAI“ kompanijos 2021 metais. Tai buvo pirmas sėkmingas matematinis modelis apjungęs galimybę modeliuoti kartu vaizdo ir teksto duomenis. Mano tikslas yra ištirti, kaip šis modelis gali būti pritaikytas laiko eilučių prognozavimui, naudojant kriptovaliutų duomenis ir „Twitter“ žinučių duomenis.

Kadangi CLIP modelis yra labai neseniai išleistas ir jis buvo sukurtas vaizdo ir teksto modeliavimui, o laiko eilučių ir teksto modeliavimui tokio modelio nėra, todėl jo nelabai eina tiesiogiai palyginti su esamais ir nusistovėjusiais modeliais. Taip pat palyginimui trūksta viešai prieinamų duomenų aibių, jos turi būti rankomis sužymėtos ir egzempliorių turi būti daug.

Bakalauro darbo tikslas - atilkti finansinių laiko eilučių ir su jomis susijusių Twitter įrašų(tekstiniu formatu) modeliavimą, identifikuojant galimybę apjungti finansines laiko eilutes ir teksto įrašų duomenis.

Bakalaurinio darbo uždaviniai:

- Kriptovaliutų kainų ir Twitter[1] įrašų duomenų surinkimas, apdorojimas.
- Duomenų jungimas paruošiant mokymo ir validavimo duomenų aibes.
- Mokslinės literatūros skaitymas, analizavimas, esamo viešo kodo analizė.
- Originalaus CLIP modelio kūrimas, parametrų vertinimas.
- Modifikuoto CLIP modelio realizavimas, parametrų vertinimas.
- Rezultatų palyginimas.
- Išvadų formulavimas.

[1] Twitter

2 Temos aprašymas

Kriptovaliutų rinka pastaraisiais metais išgyvena didelį augimą ir populiarumą, kartu su sparčiais pokyčiais ir nestabilumu. Dėl šios priežasties, efektyvi kriptovaliutų rinkos analizė ir prognozavimas yra itin svarbus tiek naujiems, tiek patyrusiems investuotojams bei mokslininkams. Šiame bakalauro darbe siekiu ištirti naujo tipo CLIP (Contrastive Language-Image Pretraining) modelio taikymo galimybes kriptovaliutų rinkos analizėje, naudojantis laiko eilučių duomenimis apie Bitcoin (BTC), Ethereum (ETH) ir Ripple (XRP) kriptovaliutas, bei „Twitter“ pranešimų duomenimis.

Kriptovaliutų rinkos analizė yra sudėtingas procesas, reikalaujantis daug skirtingų duomenų šaltinių ir sprendimų priėmimo strategijų. CLIP modelis, kaip vienas iš pažangiausių dirbtinio intelekto modelių, sujungia vaizdo ir teksto apdorojimo galimybes, todėl jis gali pasiūlyti naują požiūrį į kriptovaliutų rinkos analizę.

Šiame bakalauro darbe taip pat tyrinėsiu pakoreguotą CLIP modelio versiją kurioje vietoj paveiksliukų bus naudojama laiko eilutės.

Šio darbo tikslas yra išsiaiškinti, ar CLIP modelis gali būti naudingas kriptovaliutų rinkos pokyčių prognozavime ir analizėje, taip pat nustatyti, kaip šis modelis gali padėti investuotojams priimti informuotus sprendimus ir suprasti šio sektoriaus tendencijas. Siekiant šio tikslo, bus atliktas CLIP modelio parametrų vertinimas, naudojant surinktus laiko eilučių ir „Twitter“ pranešimų duomenis, bei modelio veiksmingumo vertinimas.

Atlikdamas šį tyrimą, tikiuosi prisidėti prie naujų technologijų, tokių kaip CLIP, taikymo kriptovaliutų rinkos analizėje ir prognozavime, bei skatinti tolesnį šio sektoriaus tyrinėjimą.

3 Pirminė duomenų analizė

3.1 Laiko eilutės

3.1.1 Duomenų šaltinis

CryptoDataDownload[2] yra duomenų šaltinis, kuris siūlo nemokamus kriptovaliutų istorinius duomenis, skirtus moksliniams tyrimams, analizei ir strategijų kūrimui. Ši platforma suteikia prieigą prie didelio kiekio kriptovaliutų prekybos duomenų, apimančių įvairias prekybos platformas, valiutų poras, intervalus ir laikotarpius. CryptoDataDownload yra svarbus šaltinis tiek pradedantiesiems, tiek pažengusiems kriptovaliutų tyrėjams ir prekiautojams, nes ji leidžia greitai ir lengvai gauti reikalingus duomenis įvairioms analizės užduotims.

CryptoDataDownload pateikia duomenis iš garsių kriptovaliutų keityk-

[2] CryptoDataDownload

lų, tokių kaip Binance[3], Bitstamp[4], Coinbase[5], Kraken[6], Bitfinex[7] ir kitų. Duomenys yra pateikiami CSV formato failuose, kurie yra lengvai importuojami į populiarius duomenų analizės įrankius, tokius kaip Python[8] pandas[9] biblioteka, R[10], MATLAB[11] arba Microsoft Excel[12].

CryptoDataDownload siūlo įvairius duomenų intervalus, pradedant nuo minutinių ir baigiant dieniniais duomenimis. Tai leidžia tyrėjams ir prekiautojams pasirinkti tinkamą duomenų rezoliuciją, atsižvelgiant į jų poreikius ir strategijas. Be to, platforma pateikia duomenis apie daugiau nei 2000 kriptovaliutų, suteikdama galimybę tyrinėti įvairius kriptovaliutų rinkos aspektus ir tendencijas.

3.1.2 Duomenų rinkimas

Šiame bakalauro darbe CryptoDataDownload duomenys yra naudojami kaip pagrindinis duomenų šaltinis modelio parametrų vertinimui. Aš pasirinkau Binance prekybos platformos duomenis, nes tai yra viena iš didžiausių ir populiariausių kriptovaliutų keityklų, suteikianti daug duomenų apie įvairias valiutų poras. Šiuo atveju mes naudojame BTCUSDT, ETHUSDT ir XRPUSDT valiutų porų duomenis, kadangi tai yra labai populiarios ir plačiai prekiaujamos poros kriptovaliutų rinkoje.

Taip pat svarbu paminėti kad duomenis galima parsisiųsti rankiniu arba automatiniu būdu naudojant populiaras programavimo kalbas.

3.1.3 Duomenų analizės įrankiai

"Pandas" yra biblioteka Python programavimo kalboje, skirta duomenų analizei ir manipuliacijai. Ši biblioteka suteikia greitą, lankstų ir aiškų būdą darbui su struktūrizuotais duomenimis. "Pandas" leidžia dirbti su įvairiais duomenų formatais, tokiais kaip CSV, Excel ir SQL duomenų bazės. Pagrindinės "Pandas" funkcijos yra duomenų struktūrų kūrimas, duomenų valymas, filtravimas, rikiavimas, grupavimas, sujungimas, ir kitos duomenų analizės operacijos.

3.1.4 Duomenų analizė

Šiame skyriuje aprašysiu metodus skirtus laiko eilučių duomenims, susijusiems su kriptovaliutų kainomis, paruošti.

[3] Binance
[4] Bitstamp
[5] Coinbase
[6] Kraken
[7] Bitfinex
[8] Python
[9] pandas
[10] R
[11] MATLAB
[12] Microsoft Excel

Kadangi šie duomenys nėra iš karto tinkami parametrų vertinimui, reikia juos paruošti. Duomenų paruošimui buvo naudojama pandas biblioteka skirta Python programavimo kalbai.

Žingsniai reikalingi laiko eilučių duomenims paruošti:

- Nuskaityti duomenų failą, praleidžiant pirmąją eilutę, kurioje yra svetainės nuoroda.
- Apgręžti eilutes, nes svetainės duomenyse naujausi įrašai yra apačioje.
- Išsaugoti tik reikalingus stulpelius: Date, Open, High, Low, Close ir Volume USDT.
- Pervadinti "Volume USDT" stulpelį į "Volume".
- Pakeisti laiko zoną į standartą jei nėra pakeista
- Išsaugoti duomenų failą CSV formatu be indeksavimo

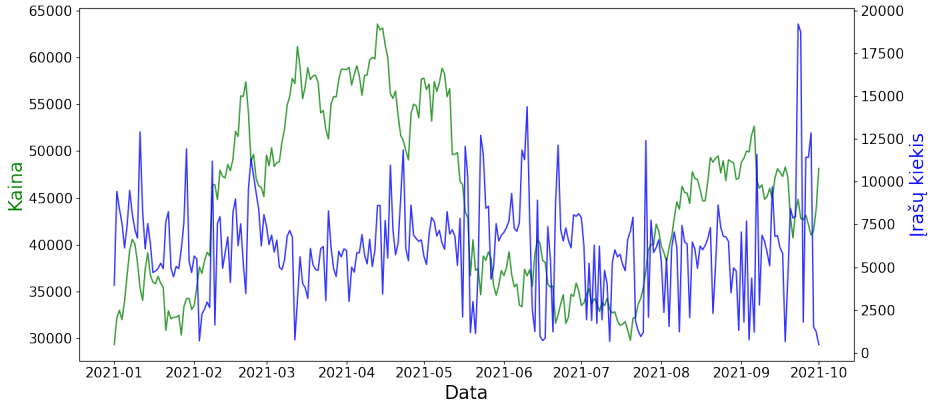
Kadangi duomenų kiekiai skiriasi skirtingo laiko dažnio duomenims, pateikiu lentelę kuri parodo kiek eilučių turi kiekvieno laiko dažnio duomenys

Laiko dažnis	Eilučių kiekis
Minutė	2001
Valanda	47894
Diena	520541

1 lentelė: Laiko eilučių duomenų kiekiai pagal laiko dažnį

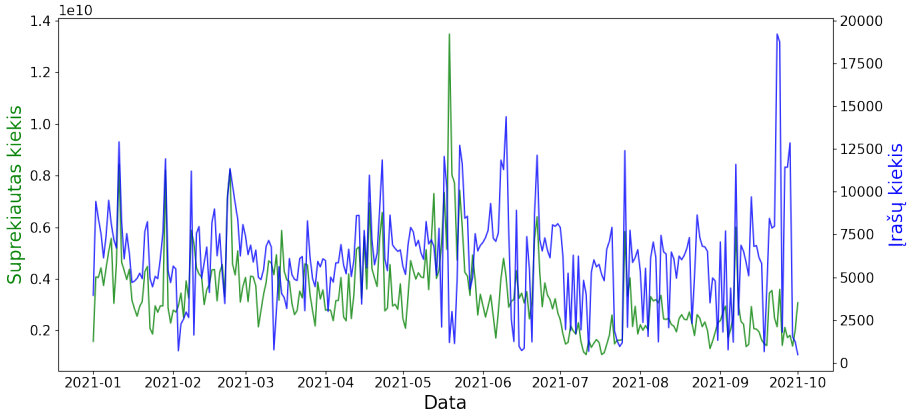
Taip pat pateikiu grafikus duomenims. Grafikai apima Bitcoin ir Ripple kriptovaliutas, tačiau nededu grafikų apie Ether kriptovaliutą nes jinai per daug glaudžiai surišta su Bitcoin visais kriterijais, tad rezultatai gaunasi beveik identiški.

1 pav.: Bitcoin kainos ir įrašų kiekio grafikas



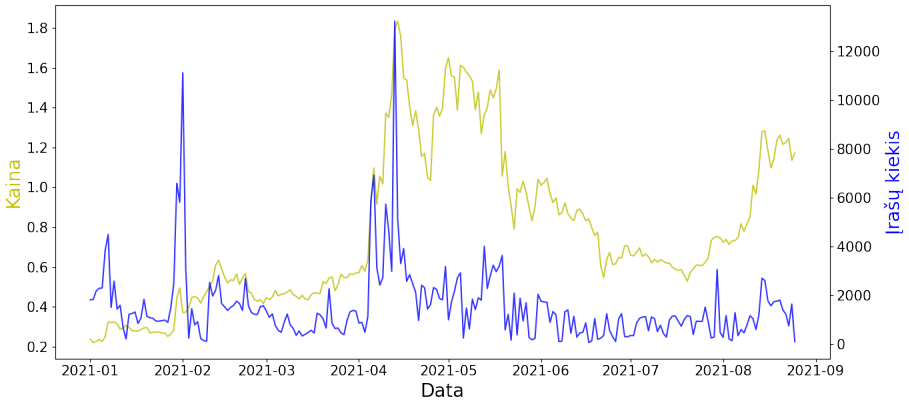
Kaip matome iš grafiko, Bitcoin kriptovaliutos kaina nelabai koreliuoja su Twitter įrašų kiekiu apie Bitcoin. Tą patvirtina ir Pearsono koreliacijos koeficientas lygus -0.01

2 pav.: Bitcoin suprekiauto kiekio ir įrašų kiekio grafikas



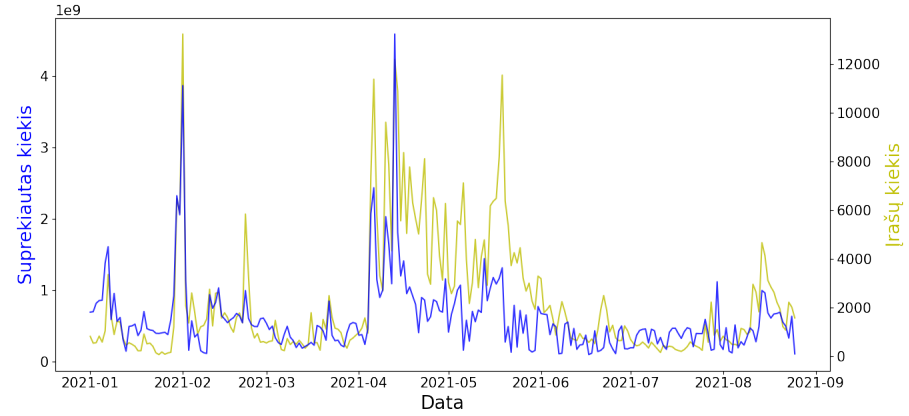
Grafike matosi kad Bitcoin suprekiauties kiekis ir Twitter įrašų kiekis stipriai varijuoja laike, tačiau tai neatrodo kaip korelacija, taip pat Pearsono koreliacijos koeficientas yra nereikšmingas (0.13)

3 pav.: Ripple kainos ir įrašų kiekio grafikas



Šiame grafike matome kad ir Ripple kriptovaliutos kaina ir Twitter įrašų kiekis apie ją buvo pasiekę maksimumą maždaug tuo pačiu laiku, 2021 metais balandžio mėnesį. Nuo to laiko abu kintamieji pradėjo mažėti, galima pastebėti menką koreliaciją, tačiau jinai silpna su Pearsono koreliacijos koeficientu 0.3

4 pav.: Ripple suprekiauto kiekio ir įrašų kiekio grafikas



Grafike aiškiai matome kad Ripple suprekiautas kiekis ir Twitter įrašų kiekis artimai susiję. Tą galima suvokti iš to kad linijos labai arti, beveik visi pakilimai sutampa ir grafikai atrodo panašus. Tą patvirtina Pearsono koreliacijos koeficiento reikšmė lygi 0.65.

3.2 Twitter įrašai

3.2.1 Duomenų šaltinis

Twint[13] yra atviro kodo Python biblioteka, skirta be jokios autentifikacijos gauti ir analizuoti „Twitter“ duomenis. Ji leidžia naudotojams lengvai rasti ir surinkti informaciją apie „Twitter“ paskyras, pranešimus, raktinius žodžius ir kitus duomenis be jokios autentifikacijos, kas reiškia, kad naudotojams nereikia prisijungti prie „Twitter“ API[14] ir gauti raktą. Pagrindinės Twint bibliotekos savybės yra:

- Pranešimų paieška pagal naudotoją, raktinį žodį, datos ribas ir kitus parametrus.
- „Twitter“ naudotojų paieška pagal raktinius žodžius, vietą, šaltinius ir kitus kriterijus.
- Gautos informacijos išsaugojimas įvairiais formatais, pvz. CSV, JSON, SQLite.
- Gautos informacijos analizė ir vizualizavimas.
- Paralelizavimo galimybės leidžia greitai surinkti didelius duomenų kiekius.

Twint yra ypač naudingas akademikams, tyrėjams, žurnalistams ir verslininkams, kurie nori greitai surinkti ir analizuoti „Twitter“ duomenis be jokių apribojimų, susijusių su „Twitter“ API. Dėl savo atviro kodo pobūdžio Twint leidžia naudotojams pritaikyti ir praplėsti jos funkcijas, kad atitiktų jų poreikius ir gauti norimą informaciją iš „Twitter“ platformos.

3.2.2 Duomenų rinkimas

Painaudodamas Twint biblioteka Python kalboje, rinkau duomenis pagal tokią konfiguraciją:

- Ieškoma įrašų su raktažodžiais "$BTC", "$ETH", "$XRP"
- Rezultate būtinai turi būti stulpeliai "language", "created_at", "tweet", "username"
- Informacija ieškoma nuo 2021-01-01 datos
- Informacija ieškoma iki 2023-02-19 datos
- Kiekvienai dienai rezultatai atskirame faile
- Rezultatai saugomi CSV tipo faile

[13] Twint
[14] Twitter API

Tačiau duomenų rinkimas su Twint turėjo savo problemų, daug kartų buvo viršytas Twitter puslapio vizitų limitas ir teko restartuoti duomenų rinkimo kodą iš naujo. Tam programos kode buvo padaryti pakeitimai kad Twint rinktų duomenis kiekvienai dienai atskirai ir saugotų visus failus atskirai, tokiu būdu jeigu programa sustos, didžiausias prarastas duomenų kiekis bus 1 diena.

3.2.3 Duomenų analizė

Iš pirmo žvilgsnio gali atrodyti kad Twitter įrašas neturi daug informacijos, tačiau kaip šiame skyriuje aptarsime, labai daug informacijos yra saugoma, tačiau nerodoma Twitter naudotojams. Ta informacija panaudojama vidiniams algoritmams ir mašininiams modeliams.

Kadangi Twitter saugo daug informacijos apie kiekvieną irašą, ji taip pat buvo surinkta ir duomenų rinkinyje.

Gauti stulpeliai:

- “id” - unikalus pranešimo identifikatorius.
- “conversation id” - unikalus pokalbio identifikatorius, kuriam priklauso pranešimas.
- “created at” - pranešimo sukūrimo data ir laikas (žymės formato).
- “date” - pranešimo sukūrimo data.
- “time” - pranešimo sukūrimo laikas.
- “timezone” - naudotojo laiko juosta.
- “user id” - unikalus naudotojo identifikatorius.
- “username” - naudotojo slapyvardis.
- “name” - naudotojo vardas.
- “place” - vieta, iš kurios buvo išsiųstas pranešimas (jei pateikiama).
- “tweet” - pranešimo tekstas.
- “language” - pranešimo kalba.
- “mentions” - pranešime minėtų naudotojų sąrašas.
- “urls” - pranešime esančių nuorodų sąrašas.
- “photos” - pranešime esančių nuotraukų sąrašas.
- “replies count” - pranešimo atsakymų skaičius.
- “retweets count” - pranešimo pasidalinimų skaičius.

- “likes count” - pranešimo patikusių reakcijų skaičius.
- “hashtags” - pranešime esančių žymių sąrašas.
- “cashtags” - pranešime esančių „cashtag“ sąrašas (pavyzdžiui, "$TSLA").
- “link” - tiesioginė nuoroda į pranešimą.
- “retweet” - rodo, ar pranešimas yra pasidalinimas (True) ar originalus pranešimas (False).
- “quote Url” - nuoroda į pranešimą, kuris yra cituojamas (jei taikoma).
- “video” - video numeris pranešime (0, jei nėra).
- “thumbnail” - nuotraukos atvaizdas, susijęs su video pranešime.
- “near” - artumas, susijęs su pranešimu (jei pateikiama).
- “geo” - geografinė padėtis, susijusi su pranešimu (jei pateikiama).
- “source” - pranešimo šaltinis (pvz., naudotojo įrenginys).
- “user rt id” - unikalus pasidalinusio naudotojo identifikatorius.
- “user rt” - pasidalinusio naudotojo slapyvardis.
- “retweet id” - unikalus pasidalinimo identifikatorius.
- “reply to” - pranešimų, į kuriuos atsakyta, sąrašas.
- “retweet date” - pranešimo pasidalinimo data ir laikas (jei taikoma).
- “translate” - pranešimo vertimas (jei pateikiama).
- “trans src” - pranešimo vertimo originali kalba
- “trans dest” - praneįimo vertimo galutinė kalba

Tačiau šis bakalauro darbas pagrinde tiria teksto ir laiko eilučių ryši, tad buvo palikti trys stulpeliai: Tweet, Username, Timestamp.

Duomenys buvo rinkti nuo datos 2021-01-01, tačiau apie skirtingas kriptovaliutas yra skirtingai kuriama įrašų tad duomenų kiekis surinktas kiekvienai kriptovaliutai skiriasi. Kiekiai pateikiami šioje lentelėje:

Valiuta	Paskutinė data	Eilučių kiekis
BTC	2021-09-30	1685865
ETH	2021-07-08	787021
XRP	2021-08-24	363441

2 lentelė: Skirtingų kriptovaliutų surinkti Twitter įrašų kiekiai

4 CLIP modelis

CLIP modelis - "OpenAI" kompanijos sukurtas dirbtinio intelekto modelis kuris sugeba susieti vaizdus ir juos apibūdinančius tekstus

Pagrindinė CLIP modelio kūrimo idėja buvo sukurti sistemą, kuri galėtų suprasti ir interpretuoti vaizdus bei tekstą kartu, panaudojant didelių duomenų rinkinių analizę. CLIP modelis mokomas atpažinti ir sujungti vaizdo ir teksto duomenis, kad galėtų generuoti tinkamus atsakymus į įvairius klausimus, apimančius tiek vaizdo, tiek teksto informaciją.

CLIP modelio parametrų vertinimo procesas yra pagrįstas kontrastingo mokymo principu. Tai reiškia, kad modelis yra mokomas atpažinti, kurie vaizdai ir tekstai yra susiję, ir atskirti juos nuo nesusijusių vaizdo ir teksto porų. Šis parametrų vertinimo metodas leidžia CLIP modeliui gerai suprasti ir interpretuoti tiek vaizdo, tiek teksto duomenis.

Vienas iš didžiausių CLIP modelio privalumų yra jo universalumas. Modelis gali būti naudojamas įvairiems tikslams, tokiems kaip vaizdo atpažinimas, teksto generavimas ar atsakymų generavimas, remiantis vaizdo ir teksto duomenimis. Be to, CLIP modelis yra labai lankstus ir gali dirbti su įvairių tipų duomenimis, todėl jis gali būti naudingas daugelyje skirtingų AI taikymo sričių.

Dažniausiai jis naudojamas vaizdo atpažinime ir teksto generavime pagal duotą paveiksliuką.

4.1 Architektūra

CLIP modelis susideda iš dviejų pagrindinių komponentų: vaizdo transformerio ir teksto transformerio. Šie du komponentai yra bendrai naudojami parametrų vertinimui su kiekviena įeitimi, leidžiant modeliui išmokti bendrą vaizdų ir teksto atvaizdavimo erdvę.

Pagrindinė CLIP modelio idėja yra mokyti šiuos du komponentus dirbti kartu, kad būtų galima susieti vaizdo ir teksto duomenis. Tam naudojamas kontrastinis parametrų vertinimas. Kontrastingo parametrų vertinimo metu modelis išmoksta atpažinti, kurie vaizdai ir tekstai yra susiję, ir atskirti juos nuo nesusijusių vaizdo ir teksto porų. Šis procesas vyksta taip, kad modelis mažina atstumą tarp susijusių vaizdo ir teksto vektorių, o didina atstumą tarp nesusijusių vaizdo ir teksto vektorių.

Galiausiai, CLIP modelis mokomas generuoti vektorių atstumus tarp vaizdo ir teksto, kurie gali būti naudojami nustatyti, koks tekstas geriausiai atitinka vaizdą, ir atvirkščiai. Tai leidžia modeliui atlikti užduotis, kurios reikalauja vaizdo ir teksto duomenų sąveikos, pvz., generuoti aprašymus vaizdams arba rasti vaizdus, atitinkančius tam tikrą tekstą.

Kadangi kai modelis mokosi jam yra paduodama 2 dalykai: vaizdas ir tekstas, jis vienu metu naudoja abu transformerius ir kai duomenys praeina pro juos, jis bando sujungti juos logiškai viduje.

Taip pat užrašau modelio matematinį pavidalą:

$$P = paveiksliukas \in \mathbb{R}^{3\times100\times100}$$

$$T = tekstas \in \mathbb{R}^{30522}$$

$$E_P = Enkoderis_P(P) \in \mathbb{R}^{256}$$

$$E_T = Enkoderis_T(T) \in \mathbb{R}^{256}$$

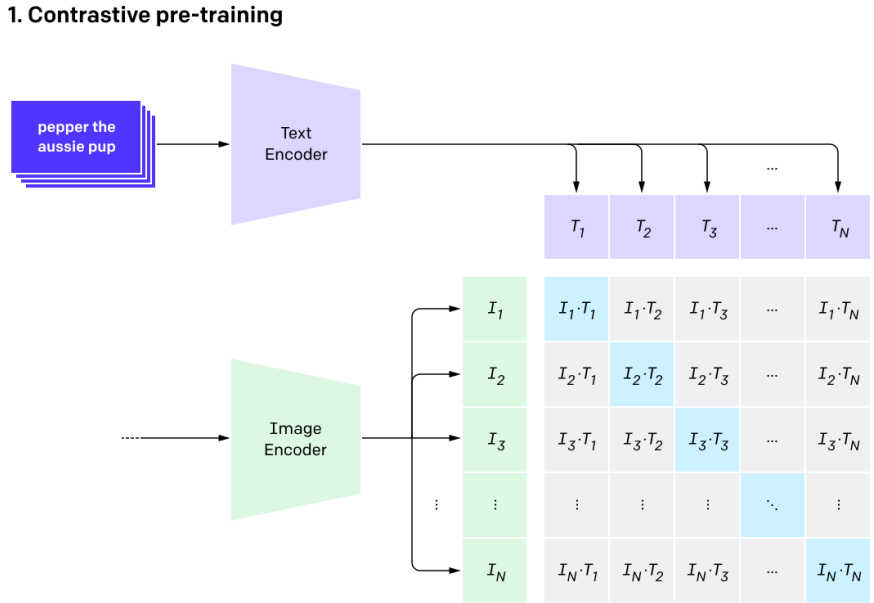
$$Nuotolis = kosinosoArtumas(E_P, E_T)$$

$$Vaizdo\_klasifikatorius = F_P(Nuotolis)$$

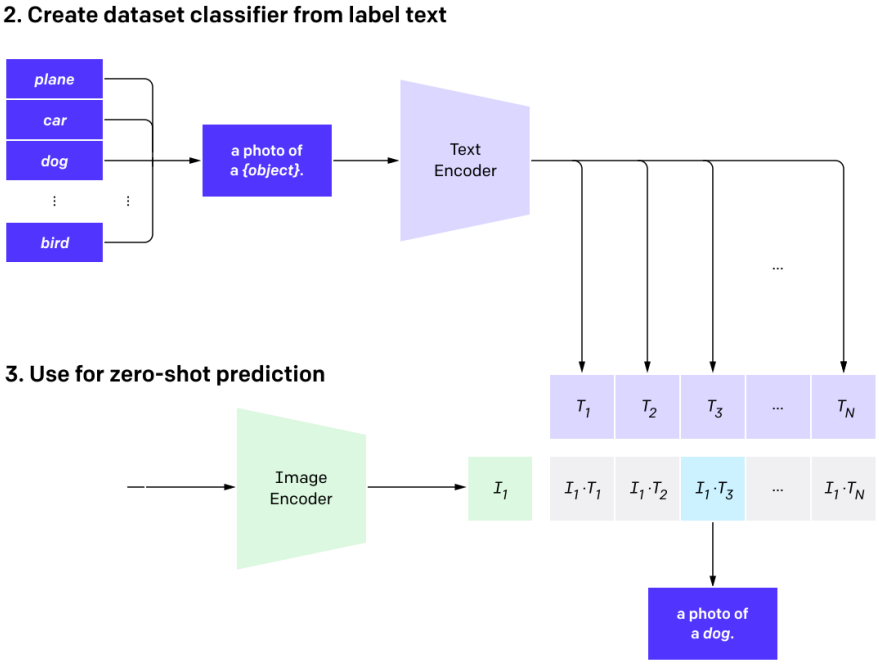
$$Teksto\_klasifikatorius = F_T(Nuotolis)$$

Tada norint klasifikuoti ar spėti tekstą, tereikia paduoti arba vaizdą, arba tekstą ir modelis sugeneruoja pagal išmoktus transformerio ir vidinius svorius.

5 pav.: CLIP modelio architektūra



6 pav.: CLIP modelio klasifikavimo architektūra



4.2 Vaizdo transformeris

Vaizdo transformeris atsakingas už vaizdų apdorojimą. Skirtingai nuo tradicinių konvoliucinių neuroninių tinklų (CNN), kurie naudoja konvoliucinius sluoksnius, vaizdo transformeris taiko transformerio architektūrą, pradinėms projekto natūralios kalbos apdorojimo užduotims. Įvesties vaizdas yra padalintas į fiksuoto dydžio, nesikertančias dalis, ir kiekviena dalis yra tiesiogiai įterpiama į plokštumos vektorių. Šie vektoriai tada perduodami per keletą transformatoriaus sluoksnių, siekiant gauti globalų vaizdo supratimą.

4.3 Teksto transformeris

Natūralios kalbos apdorojimo dalis naudoja taip pat transformerio architektūrą, skirtą teksto apdorojimui. Ji apdoroja tekstinę informaciją, naudodamas pozicinį kodavimą ir transformerio sluoksnius, leidžiančius modeliui suvokti kalbos kontekstą.

4.4 Parametrų vertinimo metodika

CLIP modelis yra išmokomas, naudojant kontrastinį parametrų vertinimo būdą. Per šį procesą modelis atlieka dvi užduotis: teigiamą teksto ir vaizdo porą susieja į vieną tašką bendroje atvaizdavimo erdvėje, tuo tarpu neigiamas poras stengiasi atitolinti. Taip modelis sukuria bendrą atvaizdavimo erdvę, kurioje vaizdai ir jų atitinkamas tekstas yra artimai susiję.

Parametrų vertinimo procesas vyksta, naudojant mini-partijų gradientinį nusileidimą ir kontrastinę entropijos funkciją. Modelio efektyvumą parametrų vertinimo metu padidina ankstyvus optimizavimas ir duomenų kiekio padidinimas.

4.5 Taikymas

Dėl savo unikalaus vaizdo ir teksto suvokimo gebėjimo, CLIP modeliai gali būti pritaikyti įvairiems uždaviniams:

- Nulinės eilės klasifikacija: Modelis gali klasifikuoti vaizdus į kategorijas, kurių iš anksto nemokė, remdamasis tik aprašymu ar etiketėmis.
- Vaizdo antraštės: Modelis gali generuoti informatyvias ir susijusias antraštes vaizdams, pagrįstas jų vizualiniu turiniu
- Vizualus klausimų atsakymas: Modelis gali atsakyti į klausimus, susijusius su vaizdo turiniu, remdamasis tiek tekstu, tiek vaizdu.

5 Eksperimentai

5.1 Originalus Clip modelis

5.1.1 Duomenų paruošimo įrankiai

"Librosa"[15] yra Python programavimo kalbos biblioteka, skirta muzikos ir garso signalų analizei. Ši biblioteka siūlo įvairias funkcijas garso įrašų įkrovimui, spektrogramų sudarymui, ypatybių išgavimui, taip pat suteikia galimybę atlikti įvairius signalų apdorojimo uždavinius, tokius kaip tempimo ir tono keitimas. Be to, ši biblioteka leidžia atlikti garso klasifikavimą ir muzikos informacijos paiešką, pavyzdžiui, automatinį akordų atpažinimą ar takto nustatymą.

"NumPy"[16] yra biblioteka Python programavimo kalboje, skirta didelėms, daugiamatėms masyvų ir matricų apdorojimui, taip pat pateikianti didelį matematinės funkcijos, kurios operuoja ant šių masyvų, sąrašą. Tai yra viena iš pagrindinių bibliotekų duomenų mokslui ir mašininiam mokymuisi Python kalboje dėl jos efektyvumo ir lankstumo dirbant su dideliais duomenų kiekiais. Biblioteka taip pat leidžia atlikti kompleksinius matematinius skaičiavimus ir yra pagrindas kitoms svarbioms Python bibliotekoms, tokios kaip Pandas, Matplotlib ir Scipy.

"Matplotlib"[17] yra biblioteka Python programavimo kalboje, kuri yra plačiai naudojama grafikų ir diagramų kūrimui. Ji gali generuoti linijines diagramas, stulpelių diagramas, histogramas, dispersijos diagramas ir kitus vi-

[15] Librosa
[16] Numpy
[17] Matplotlib

zualizacijos tipus su mažai eilučių kodo. Be to, ji suteikia galingas personalizavimo galimybes, leidžia reguliuoti spalvas, stilius, antraštes, ašis ir kitus elementus. Dėl jos lankstumo ir funkcionalumo, "Matplotlib" yra vienas iš pagrindinių įrankių duomenų analizei ir moksliniam programavimui Python kalboje.

"tqdm"[18] yra Python programavimo kalbos biblioteka, kuri yra naudojama progreso juostos funkcionalumui realizuoti. Šis įrankis gali būti naudojamas kodo blokuose, kurie užtrunka ilgą laiką vykdyti, pavyzdžiui, ciklai arba ilgai trunkantys skaičiavimai. Ši biblioteka suteikia vizualų atvaizdavimą, kaip greitai kodas vyksta ir kiek laiko tikėtis, kol jis bus baigtas. Biblioteka yra lengvai integruojama ir lanksti, galinti veikti su konsole, grafinėmis sąsajomis ir Python failais.

5.1.2 Duomenų paruošimas

Duomenų paruošimui buvo naudojau "pandas", "librosa", "numpy", "matplotlib" ir "tqdm" bibliotekas. Pagrindinis duomenų paruošimo tikslas buvo paversti laiko eilučių duomenis į paveiksliukus, kad būtų galima naudoti gautus paveikslėlius kaip mokymo duomenų rinkinį pradinio CLIP modelio parametrų vertinimui. Naudojau chromogramos transformaciją, kuri gali bet kokią laiko eilutę atvaizduoti kaip paveikslėlį su norimais parametrais.

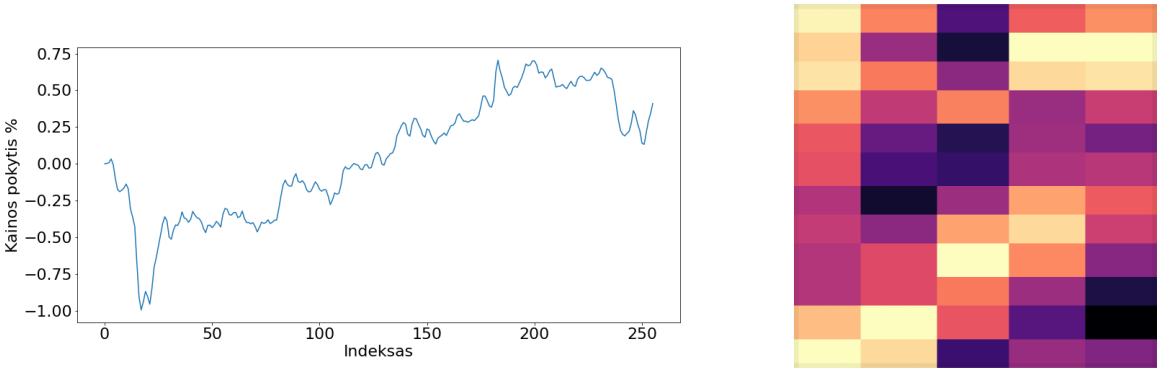
Generuojant paveiksliukus buvo naudojami keli specialūs nurodymai:

- Paveikslėlio plots ir ilgis yra 1 colis
- Nuimamos ašys ir legenda
- Paveikslėlio kraštai prispaudžiami prie turinio
- Failas saugomas PNG formatu

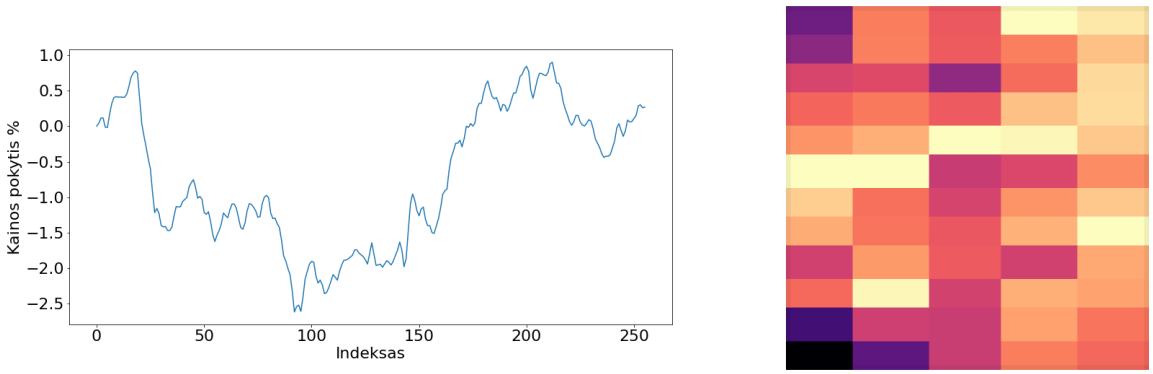
Su sukurtu Python kodu buvo sugeneruota 10000 paveiksliukų, pateikiu pavyzdžius.

[18] tqdm

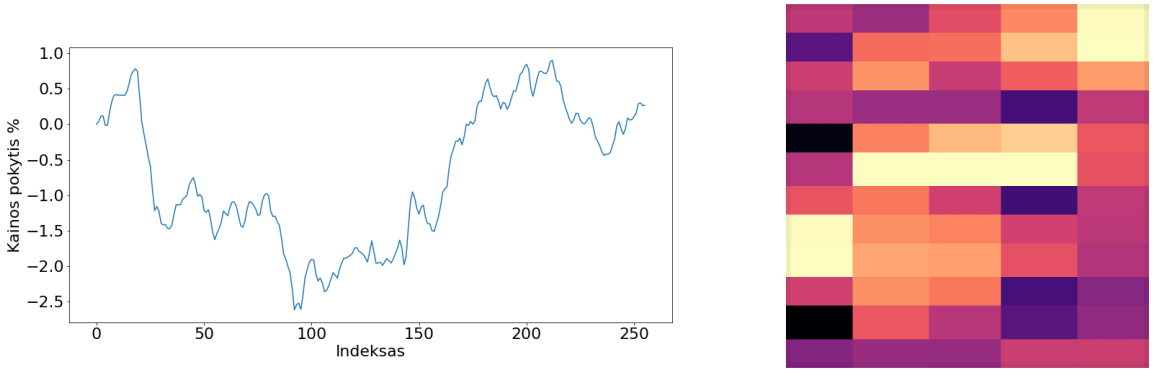
7 pav.: Pirmas pavyzdys konvertuotos laiko eilutės, atstovauja tekstą "$CNS joining #BinanceSmartChain This week. Buy now and convert to $CNR for hourly yields"



8 pav.: Antras pavyzdys konvertuotos laiko eilutės, atstovauja tekstą "What a start to the week for $ETH. Price is up against daily resistance here, so could see a pull back, but when bulls break through - should be a clear shot to $3k"



9 pav.: Trečias pavyzdys konvertuotos laiko eilutės, atsotvaują tekstą "$XRP - that's me when I think about my XRP"



5.1.3 Aprašymas

Pirmasis eksperimentas yra susijęs su laiko serijų konvertavimu į paveikslėlius naudojant "librosa" biblioteką ir toliau naudojant gautus paveikslėlius kaip mokymo duomenų rinkinį pradinio "CLIP" modelio parametrų vertinimui. Tai yra naujoviškas požiūris, kurio tikslas yra sukurti ryšį tarp skirtingų modalumo duomenų ir pagerinti modelio gebėjimą atpažinti vaizdus ir tekstus.

Eksperimento metu pirmiausia yra konvertuojamos laiko serijos į paveikslėlius naudojant "librosa" biblioteką. Tai daroma siekiant transformuoti laiko serijas į formą, kuri gali būti naudojama kaip mokymo duomenų rinkinys originalaus "CLIP" modelio parametrų vertinimui. "Librosa" biblioteka yra naudojama garsų apdorojimui, bet ji taip pat gali būti pritaikyta signalų apdorojimui, pvz., konvertuojant laiko serijas į paveikslėlius.

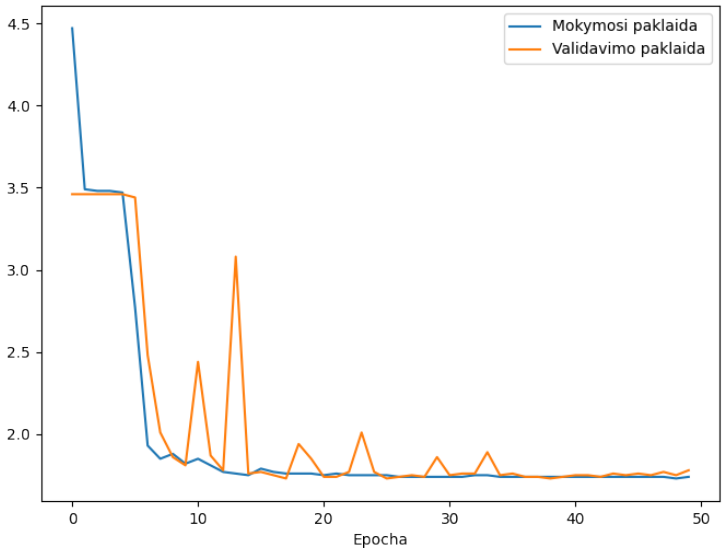
Tada gauti paveikslėliai yra naudojami kaip mokymo duomenų rinkinys pradinio "CLIP" modelio mokymui. "CLIP" yra dviejų modalumų neuroninis tinklas, kuris yra skirtas atpažinti vaizdus ir tekstus. Eksperimento metu mokymo duomenų rinkinys yra sukurtas iš paveikslėlių, gautų iš laiko serijų, ir jų susijusių teksto aprašymų. Tada šis duomenų rinkinys yra naudojamas parametrų vertinimui.

Eksperimentas Nr. 1 yra inovatyvus, nes siekia sukurti ryšį tarp skirtingų tipų duomenų ir pagerinti modelio gebėjimą atpažinti vaizdus ir tekstus. Tai gali turėti įtakos daugeliui skirtingų sričių, tokios kaip finansai, prekyba, medicina ir kt., kur reikalingas aukštas tikslumas duomenų analizėje ir prognozavime

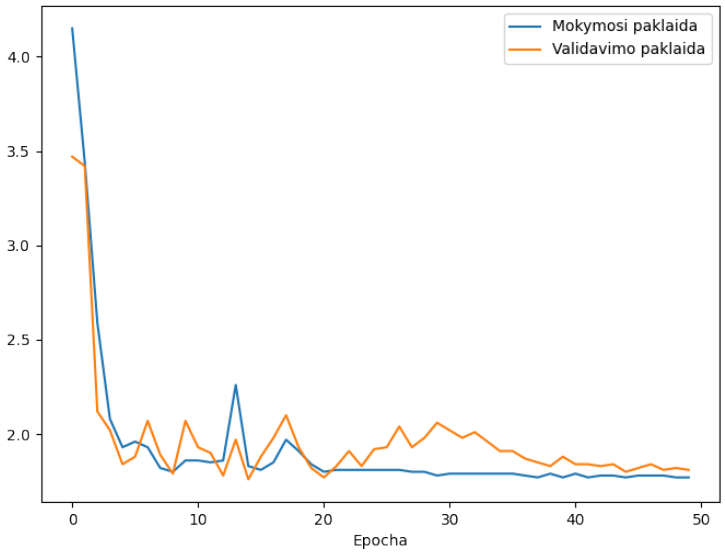
5.1.4 Rezultatai

Šiame skyriuje pateikiu originalaus CLIP modelio mokymosi rezultatus. Prieš parametrų vertinimą duomenys buvo išskirti į mokymo (75%) ir validavimo (25%) aibes. Šie grafikai buvo pagaminti įvertinus modelio parametrus po 50 epochų. Svarbu paminėti kad parametrų vertinimo procese vyksta 2 dalykai vienu metu: pagal paveikslėli modelis bando spėti atitinkamą teksto įrašą, ir pagal konkretu teksto įrašą spėti kokia laiko eilutė atitinka labiausiai.

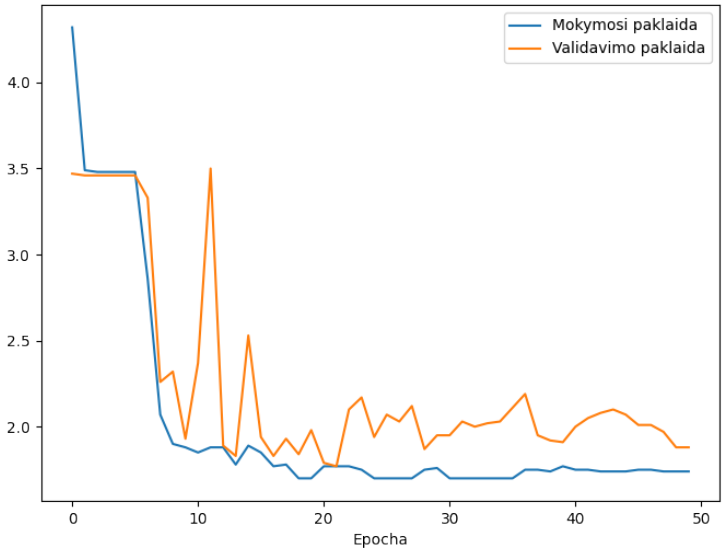
10 pav.: BTC Originalaus modelio parametrų vertinimo statistika



11 pav.: ETH Originalaus modelio parametrų vertinimo statistika



12 pav.: XRP Originalaus modelio parametrų vertinimo statistika



Kaip matome iš visų trijų grafikų, modelio validavimo ir mokymosi paklaidos nustoja mažėti po maždaug 10 epochų. Tai gali reikšti kad, pavyzdžiui modelis kiek įmanoma įvertino parametrus ir iš papildomo parametrų vertinimo naudos nebegauna. Tačiau taip pat matosi kad galutinė paklaida tik 2 kartus mažesnė nei pradinė. Tai gali reikšti kad prie duomenų paruošimo

arba modelio konstravimo reiktų dar padirbėti. BTC ir ETH grafikuose validavimo paklaida link galo nusistovi ir stipriai nebekinta, tačiau su XRP yra kita situacija, jos validavimo paklaida nuolatos stipriai kinta, kaip ir mokymo paklaida.

5.1.5 Prognozės

Kad galėčiau patikrinti ar modelis sugeneruoja skirtingas laiko eilutes skirtingo sentimento tekstams, pasinaudosiu šiais sakiniais. Vienas yra aiškiai teigiamo sentimento, kitas neigiamo.

Teigiamo sentimento sakinys: "The steady rise in BTC price not only reflects its immense value as a digital asset, but also offers a promising outlook for investors, fueling excitement and confidence in its future trajectory.

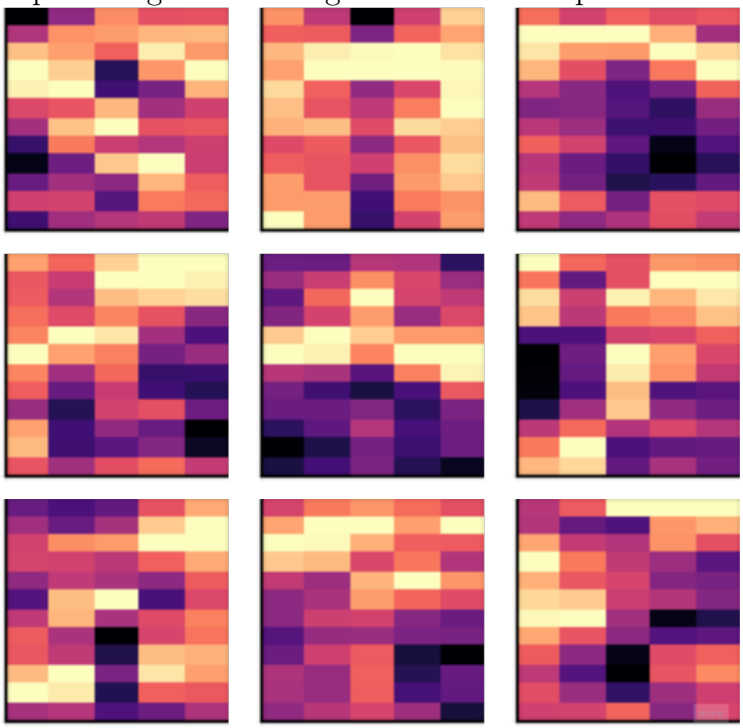
Neigiamo sentimento sakinys: "The volatile nature of BTC price can make it challenging for investors to predict and navigate, causing anxiety and uncertainty in the market."

Tada modelis sugeneravo 9 paveikslėlius kiekvienam sakiniui. Tuos paveikslėlius tada reikėjo atversti atgal į laiko eilutes ir jas sudėti į grafiką.

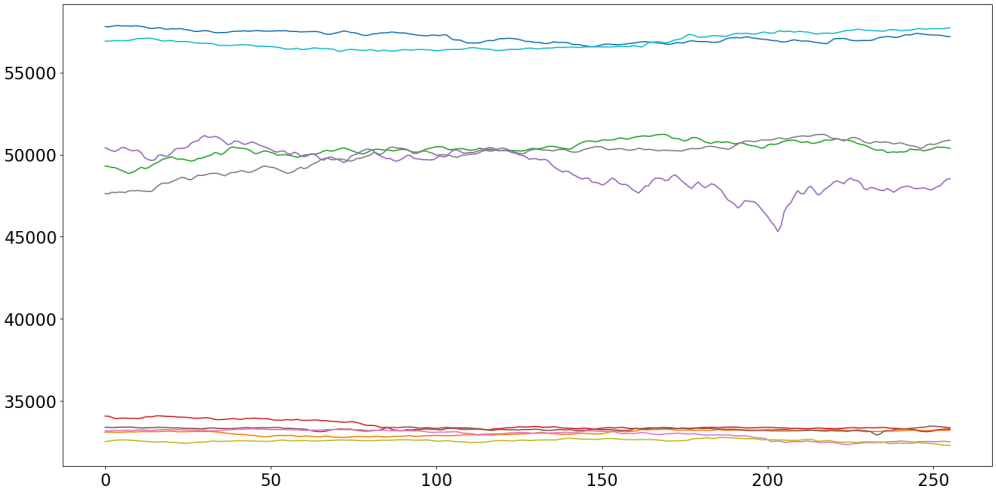
Svarbu paminėti, kad sugeneruotos teigiamo sentimento laiko eilutės bendras vidurkis buvo 46760, o neigiamos - 41329. Iš to galima spekuliuoti kad modelis viduje suprato pagal sentimentą kad teigiamo sentimentas vidutiniškai lemia didesnes kains.

Taip pat visuose originalaus CLIP modelio laiko eilučių grafikuose, rezultatai pateikiamo ne kainos pokyčiu o pačia kaina doleriais. Ir x ašyje yra indeksas, jo maksimumas yra 256 nes buvo naudojamos tokios dimensijos laiko eilutės.

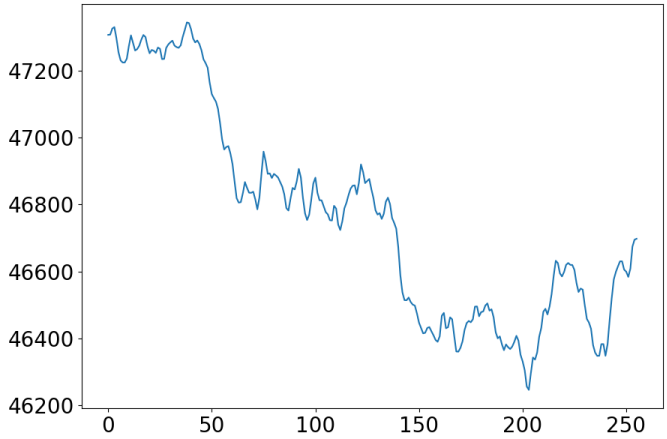
13 pav.: Sugeneruoto teigiamo sentimento paveikslėliai



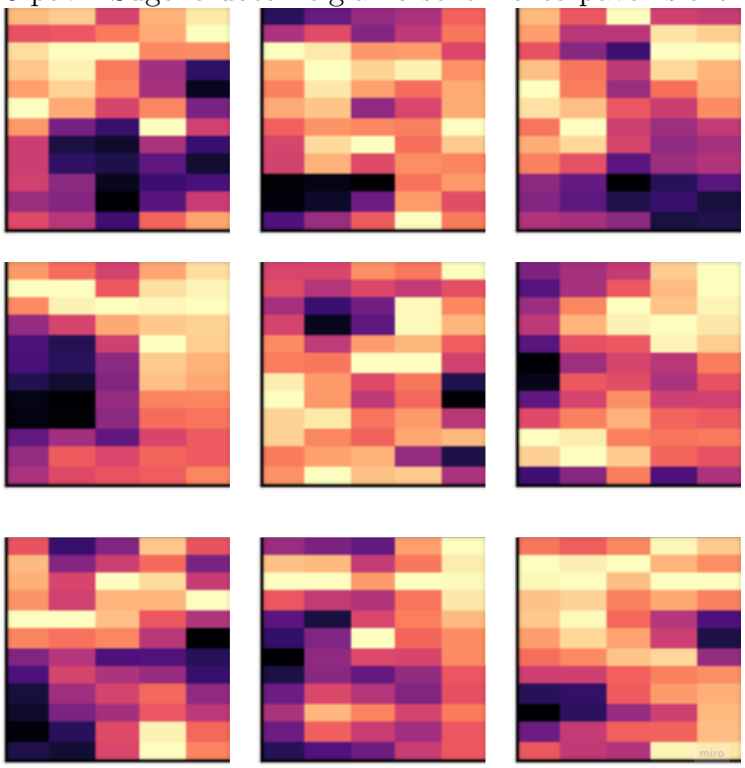
14 pav.: Sugeneruoto teigiamo sentimento laiko eilutės



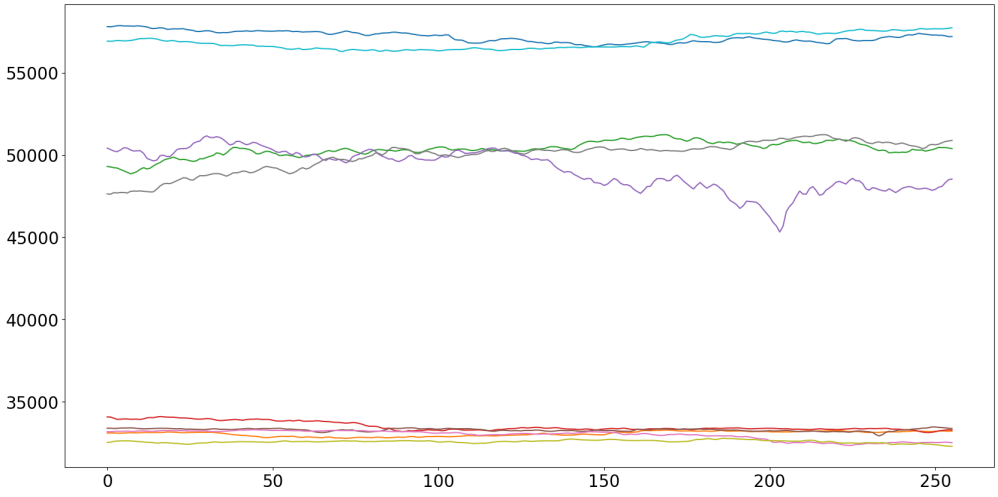
15 pav.: Teigiamo sentimento suvidurkinta laiko eilutė



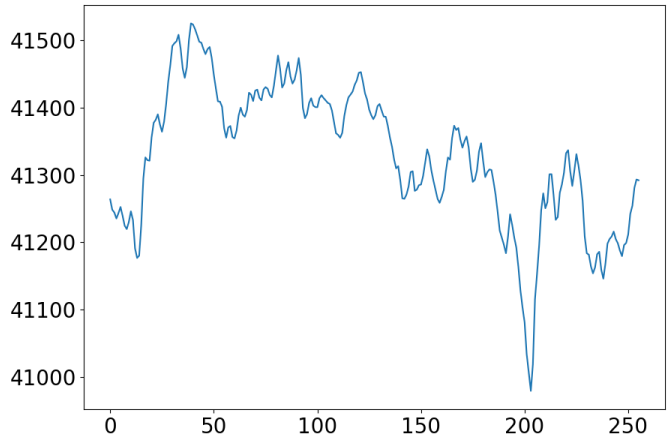
16 pav.: Sugeneruoto neigiamo sentimento paveikslėliai



17 pav.: Sugeneruoto neigiamo sentimento laiko eilutės



18 pav.: Neigiamo sentimento suvidurkinta laiko eilutė



5.2 Modifikuotas Clip modelis

Prieš tai aptariau standartinį CLIP modelį, šiame skyriuje aptarsiu kaip aš jį modifikavau kad veiktų su laiko eilutėmis.

Kadangi paveikslėlis yra tiesiog pikselių rinkinys dvimatėje matricoje, tai reiškias modelis gali priimti bet ką kas yra matrica ir vertinti parametrus. Tuo pasinaudodamas aš pakeičiau paveiksliukų įvestį kuri buvo 256x256 formos, į 256x1 dimensijos aibę.

Taip gavau modelį kuris priima laiko eilutes vietoj paveiksliukų ir atlikau ta patį parametrų vertinimo procesą.

Pateikiu modifikuoto modelio matematinę išraišką:

$$L = laiko\_eilute \in \mathbb{R}^{256}$$

$$T = tekstas \in \mathbb{R}^{30522}$$

$$E_L = Enkoderis_L(L) \in \mathbb{R}^{256}$$

$$E_T = Enkoderis_T(T) \in \mathbb{R}^{256}$$

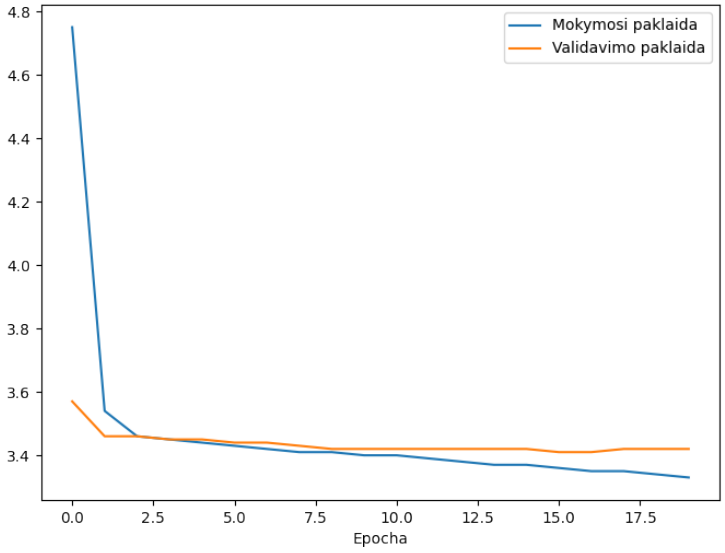
$$Nuotolis = kosinosoArtumas(E_L, E_T)$$

$$Laiko\_eilutes\_klasifikatorius = F_L(Nuotolis)$$

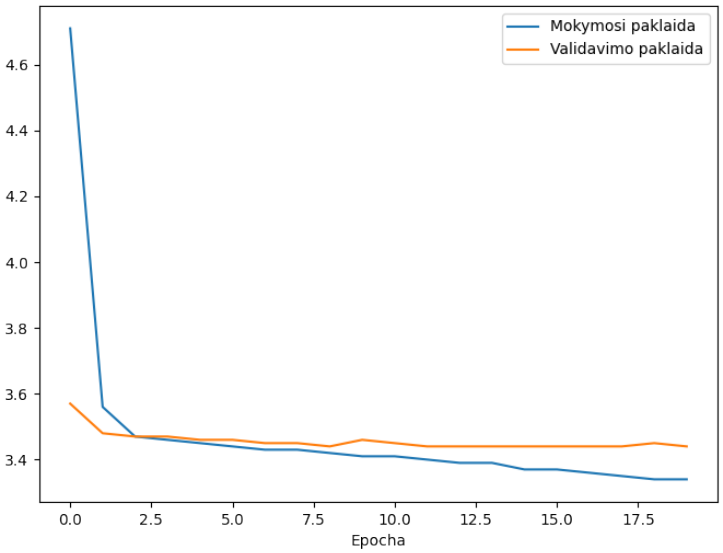
$$Teksto\_klasifikatorius = F_T(Nuotolis)$$

5.2.1 Rezultatai

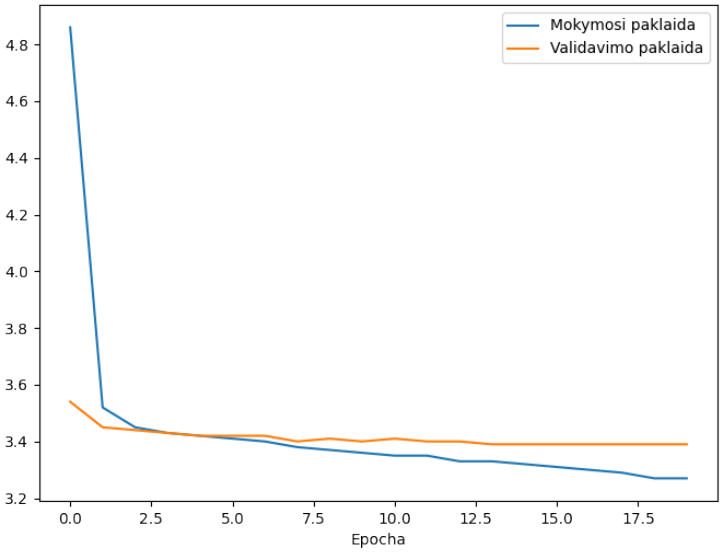
19 pav.: BTC Modifikuoto modelio parametrų vertinimo statistika



20 pav.: ETH Modifikuoto modelio parametrų vertinimo statistika



21 pav.: XRP Modifikuoto modelio parametrų vertinimo statistika



Iš grafikų matyti kad situacija su modifikuotu CLIP šiek tiek kitokia. Visi grafikai gavosi labai panašūs, skiriasi tik tais menkai kai kuriose vietose.

Validavimo paklaida labai mažai pakinta per visą 50 epochų mokymosi laikotarpį. Tai figuruoja kad CLIP modelis galimai nėra geras variantas laiko eilučių ir teksto apie jas modeliavimui.

Taip pat iš didelio mokymosi paklaidos mažėjimo, galima pamatyti kad tai rodo modelio persimokymą, tai įvyksta kai modelis per daug prisitaiko prie mokymo aibėje esančių duomenų ir prastai spėja validavimo aibės duomenis, tokiu atveju mokymosi paklaida nuolatos mažėja, o validavimo - ne.

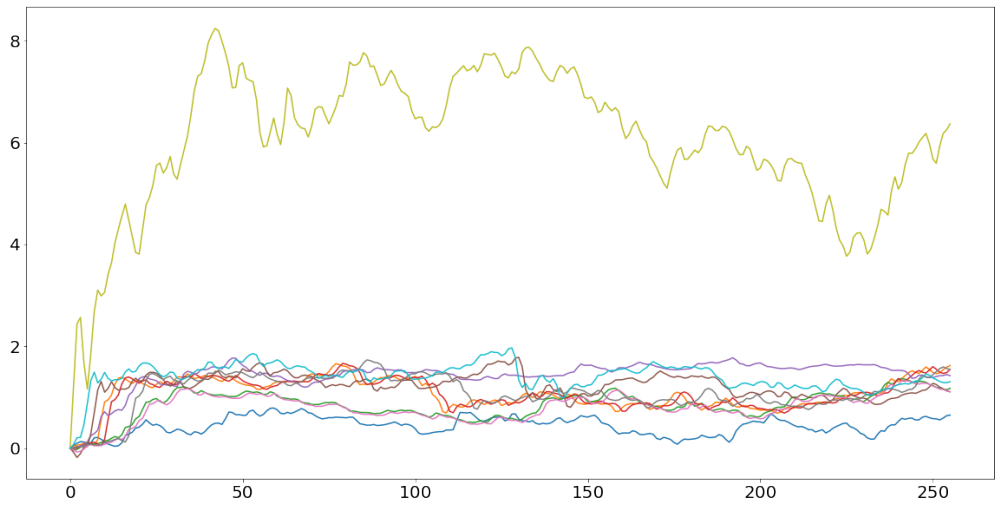
Taip pat priminsiu kad modifikuoto CLIP modelio prognozėse laiko eilutės matuojamos procentiniu pokyčiu nuo teksto įrašo publikavimo datos.

5.2.2 Prognozės

Kadangi visų kriptovaliutų modeliai atrodo panašūs iš mokymosi statistikų. Pabandysiu pažiūrėti ką modelis sugeneruos pagal skirtingo sentimento tekstą.

Su sakiniu "The steady rise in BTC price not only reflects its immense value as a digital asset, but also offers a promising outlook for investors, fueling excitement and confidence in its future trajectory.", kuris turi aiškų teigiamą sentimentą, sugeneravus 10 laiko eilučių gauname tokį grafiką:

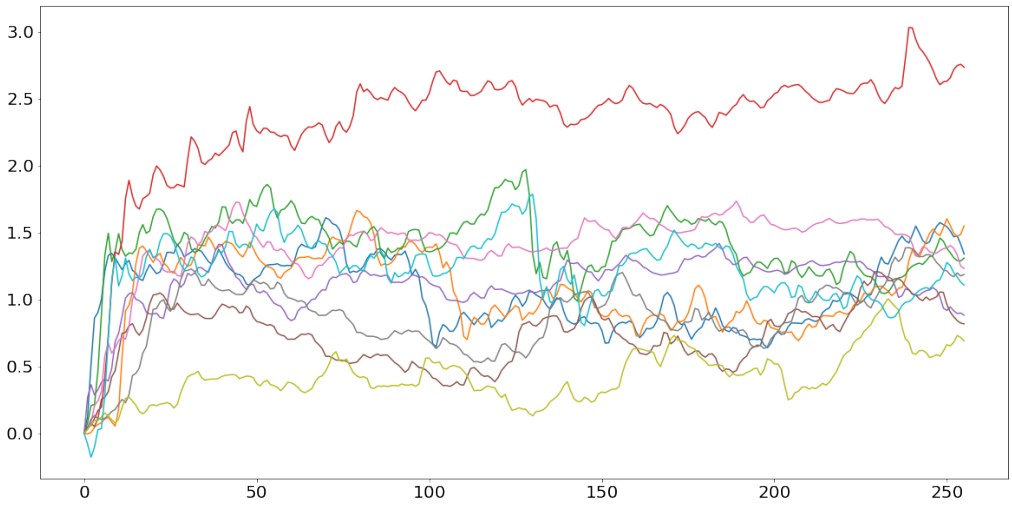
22 pav.: Sugeneruotos teigiamo sentimento laiko eilutės



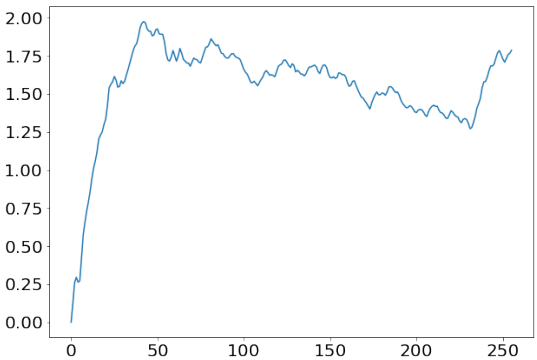
Palyginimui reiktų ir neigiamo sentimento sakinio rezultato, tad su sakiniu "The volatile nature of BTC price can make it challenging for investors to predict and navigate, causing anxiety and uncertainty in the market.", vėl generavau 100 laiko eilučių, pateikiu rezultatus.

Matome kad abiejuose rezultatuos yra panašių laiko eilučių, bet kad geriau matyti skirtumą galime suvidurkinti eilutes ir pažiūrėti kaip atrodo skirtingo sentimento eilutės tada.

23 pav.: Sugeneruotos neigiamo sentimento laiko eilutės

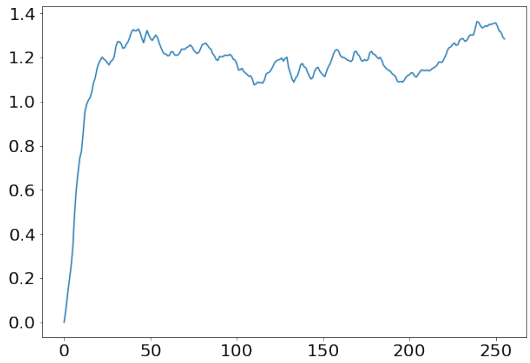


24 pav.: Teigiamo sentimento suvidurkinta laiko eilutė



Suvidurkinus teigiamo sentimento eilutė dar kartą gauname vidurkį 1.54, tačiau neigiamos eilutės atveju gauname 1.16, nors neigiamo sentimento eilutės rezultatai nėra tokie, kad kaina krenta, tačiau jinai mažesnė už teigiamo sentimento vidurkį.

25 pav.: Neigiamo sentimento suvidurkinta laiko eilutė



6 Rezultatai

Buvo atlikti 2 eksperimentai, kuriuose įvertinti parametrai 6 modeliams, nes turėjau 3 kriptovaliutas: BTC, ETH, XRP ir kiekvienam modelio ir kriptovaliutos porai įvertinau parametrus.

Pagal mokymosi ir validavimo paklaidų grafikus galima suprasti kad CLIP modelis daug informacijos apie nagrinėtus 2 duomenų tipus nesuprato.

Taip pat pastebėjau kad sugeneruotų laiko eilučių vidurkiai stipriai skiriasi priklausant ar generuojama pagal teigiamo ar neigiamo sentimento sakinius, teigiamo sentimento sakiniai sugeneravo laiko eilutes kurių vidurkiai buvo aukštesni negu neigiamo.

7 Išvados

Šis darbas prasidėjo nuo kriptovaliutų kainų ir Twitter įrašų duomenų surinkimo. Tada duomenims buvo pritaikyti valymo, apibendrinimo ir jungimo procedūros. Buvo atlikti 2 eksperimentai, pirmasis susijęs su originalaus CLIP modelio parametrų vertinimu, kitas su modifikuoto CLIP modelio parametrų vertinimu.

Gauti modeliai kaip ir visi tarpiniai rezultatai buvo saugomi ir pasibaigus mokymo procesui jie buvo panaudojami spėjimams atlikti.

Originalus CLIP modelis pagal duodamą tekstą spėja norimą kiekį paveiksliukų, tuomet norint gaut laiko eilutes reikia atlikti atvirkštinę chromogramos transformaciją.

Modifikuotas CLIP modelis iškart spėja laiko eilutes ir papildomų žingsnių nereikėjo atlikti.

Tačiau verta paminėti kad šis uždavinys buvo ir vis dar yra sunkus šiuolaikiniams mašininio mokymosi modeliams.

Iš padarytų eksperimentų galima šiek tiek suprasti kad CLIP modelis nėra geras būdas modeliuoti laiko eilutes ir Twitter įrašus apie jas.

Literatūra

[Ayd22] Saadettin Aydin. Time series analysis and some applications in medical research. *Journal of Mathematics and Statistics Studies*, 3:31–36, 10 2022.

[Cra65] J. M. Craddock. The analysis of meteorological time series for use in forecasting. *Journal of the Royal Statistical Society. Series D (The Statistician)*, 15(2):167–190, 1965.

[RKH+21] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision. *CoRR*, abs/2103.00020, 2021.

[SGO20] Omer Sezer, Ugur Gudelek, and Murat Ozbayoglu. Financial time series forecasting with deep learning : A systematic literature review: 2005–2019. *Applied Soft Computing*, 90:106181, 02 2020.

[ZPHW13] Ruijin Zhou, Yiqun Pan, Zhizhong Huang, and Qiujian Wang. Building energy use prediction using time series analysis. pages 309–313, 12 2013.

8 Priedai

```
******************
**CLIP originalus**
******************

import os
import cv2
import
timm
import torch
import itertools
import numpy as np
import pandas as pd
import torch.nn as
nn
import albumentations as A
import matplotlib.pyplot as plt
from tqdm.autonotebook import
tqdm
from transformers import DistilBertModel, DistilBertConfig, DistilBertTokenizer
import
torch
from torch import nn
import torch.nn.functional as F

class CFG:
    debug = False

image_path = "/content/drive/MyDrive/BACHELOR'S DATA/librosa-images"

captions_path = "."
    batch_size = 32
    num_workers = 4
    head_lr = 1e-3

image_encoder_lr = 1e-3
    text_encoder_lr = 1e-3
    weight_decay = 1e-3
    patience = 1

factor = 0.8
    epochs = 100
    device = torch.device("cuda" if
torch.cuda.is_available() else "cpu")

    model_name = 'resnet50'

image_embedding = 2048
    text_encoder_model = "distilbert-base-uncased"

text_embedding = 768
    text_tokenizer = "distilbert-base-uncased"
    max_length =
200

    pretrained = True # for both image encoder and text encoder
    trainable = True # for
both image encoder and text encoder
    temperature = 1.0

    # image size
    size = 224


# for projection head; used for both image and text encoders
    num_projection_layers = 1

projection_dim = 256
    dropout = 0.1

class AvgMeter:
    def __init__(self,
name="Metric"):
        self.name = name
        self.reset()
```

```
    def reset(self):

      self.avg, self.sum, self.count = [0] * 3

    def update(self, val, count=1):

self.count += count
        self.sum += val * count
        self.avg = self.sum / self.count


  def __repr__(self):
        text = f"{self.name}: {self.avg:.4f}"
        return
text

def get_lr(optimizer):
    for param_group in optimizer.param_groups:
        return
param_group["lr"]


class CLIPDataset(torch.utils.data.Dataset):
    def
__init__(self, image_filenames, captions, tokenizer, transforms):
        """

     image_filenames and cpations must have the same length; so, if there are
        multiple
captions for each image, the image_filenames must have repetitive
        file names

"""

        self.image_filenames = image_filenames
        self.captions =
list(captions)
        self.encoded_captions = tokenizer(
            list(captions),
padding=True, truncation=True, max_length=CFG.max_length
        )
        self.transforms =
transforms

    def __getitem__(self, idx):
        item = {
            key:
torch.tensor(values[idx])
            for key, values in self.encoded_captions.items()

}

        image = cv2.imread(f"{CFG.image_path}/{self.image_filenames[idx]}")

 image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
        image =
self.transforms(image=image)['image']
        item['image'] = torch.tensor(image).permute(2, 0,
1).float()
        item['caption'] = self.captions[idx]

        return item


    def
__len__(self):
        return len(self.captions)



def
get_transforms(mode="train"):
    if mode == "train":
        return
```

```
A.Compose(
            [
                A.Resize(CFG.size, CFG.size, always_apply=True),

          A.Normalize(max_pixel_value=255.0, always_apply=True),
            ]
        )

else:
        return A.Compose(
            [
                A.Resize(CFG.size, CFG.size,
always_apply=True),
                A.Normalize(max_pixel_value=255.0, always_apply=True),

       ]
        )

class ImageEncoder(nn.Module):
    """
    Encode
images to a fixed size vector
    """

    def __init__(
        self,
model_name=CFG.model_name, pretrained=CFG.pretrained, trainable=CFG.trainable
    ):

super().__init__()
        self.model = timm.create_model(
            model_name, pretrained,
num_classes=0, global_pool="avg"
        )
        for p in self.model.parameters():

           p.requires_grad = trainable

    def forward(self, x):
        return self.model(x)


class TextEncoder(nn.Module):
    def __init__(self, model_name=CFG.text_encoder_model,
pretrained=CFG.pretrained, trainable=CFG.trainable):
        super().__init__()
        if
pretrained:
            self.model = DistilBertModel.from_pretrained(model_name)
        else:

           self.model = DistilBertModel(config=DistilBertConfig())

        for p
in self.model.parameters():
            p.requires_grad = trainable

        # we are using the
CLS token hidden representation as the sentence's embedding
        self.target_token_idx = 0


   def forward(self, input_ids, attention_mask):
        output =
self.model(input_ids=input_ids, attention_mask=attention_mask)
        last_hidden_state =
output.last_hidden_state
        return last_hidden_state[:, self.target_token_idx, :]


class ProjectionHead(nn.Module):
    def __init__(
        self,
        embedding_dim,
```

```
 projection_dim=CFG.projection_dim,
        dropout=CFG.dropout
    ):

super().__init__()
        self.projection = nn.Linear(embedding_dim, projection_dim)

self.gelu = nn.GELU()
        self.fc = nn.Linear(projection_dim, projection_dim)

self.dropout = nn.Dropout(dropout)
        self.layer_norm = nn.LayerNorm(projection_dim)


   def forward(self, x):
        projected = self.projection(x)
        x =
self.gelu(projected)
        x = self.fc(x)
        x = self.dropout(x)
        x = x +
projected
        x = self.layer_norm(x)
        return x


class CLIPModel(nn.Module):

def __init__(
        self,
        temperature=CFG.temperature,

image_embedding=CFG.image_embedding,
        text_embedding=CFG.text_embedding,
    ):

super().__init__()
        self.image_encoder = ImageEncoder()
        self.text_encoder =
TextEncoder()
        self.image_projection = ProjectionHead(embedding_dim=image_embedding)

    self.text_projection = ProjectionHead(embedding_dim=text_embedding)

self.temperature = temperature

    def forward(self, batch):
        # Getting Image and Text
Features
        image_features = self.image_encoder(batch["image"])

text_features = self.text_encoder(
            input_ids=batch["input_ids"],
attention_mask=batch["attention_mask"]
        )
        # Getting Image and Text
Embeddings (with same dimension)
        image_embeddings =
self.image_projection(image_features)
        text_embeddings =
self.text_projection(text_features)

        # Calculating the Loss
        logits =
(text_embeddings @ image_embeddings.T) / self.temperature
        images_similarity =
image_embeddings @ image_embeddings.T
        texts_similarity = text_embeddings @
text_embeddings.T
        targets = F.softmax(
            (images_similarity +
texts_similarity) / 2 * self.temperature, dim=-1
        )
        texts_loss =
cross_entropy(logits, targets, reduction='none')
```

```
        images_loss = cross_entropy(logits.T,
targets.T, reduction='none')
        loss =  (images_loss + texts_loss) / 2.0 # shape:
(batch_size)
        return loss.mean()


def cross_entropy(preds, targets, reduction='none'):

   log_softmax = nn.LogSoftmax(dim=-1)
    loss = (-targets * log_softmax(preds)).sum(1)
    if
reduction == "none":
        return loss
    elif reduction == "mean":

   return loss.mean()

def make_train_valid_dfs():
    dataframe =
pd.read_csv(f"/content/drive/MyDrive/BACHELOR'S DATA/captions.csv")
    max_id =
dataframe["id"].max() + 1 if not CFG.debug else 100
    image_ids = np.arange(0,
max_id)
    np.random.seed(42)
    valid_ids = np.random.choice(
        image_ids,
size=int(0.2 * len(image_ids)), replace=False
    )
    train_ids = [id_ for id_ in image_ids
if id_ not in valid_ids]
    train_dataframe =
dataframe[dataframe["id"].isin(train_ids)].reset_index(drop=True)
    valid_dataframe
= dataframe[dataframe["id"].isin(valid_ids)].reset_index(drop=True)
    return
train_dataframe, valid_dataframe


def build_loaders(dataframe, tokenizer, mode):

transforms = get_transforms(mode=mode)
    dataset = CLIPDataset(

dataframe["image"].values,
        dataframe["caption"].values,

tokenizer=tokenizer,
        transforms=transforms,
    )
    dataloader =
torch.utils.data.DataLoader(
        dataset,
        batch_size=CFG.batch_size,

num_workers=CFG.num_workers,
        shuffle=True if mode == "train" else False,

)
    return dataloader

def train_epoch(model, train_loader, optimizer, lr_scheduler, step):

  loss_meter = AvgMeter()
    tqdm_object = tqdm(train_loader, total=len(train_loader))
    for
batch in tqdm_object:
        batch = {k: v.to(CFG.device) for k, v in batch.items() if k !=
"caption"}
        loss = model(batch)
        optimizer.zero_grad()

loss.backward()
        optimizer.step()
```

```
        if step == "batch":

lr_scheduler.step()

        count = batch["image"].size(0)

loss_meter.update(loss.item(), count)


tqdm_object.set_postfix(train_loss=loss_meter.avg, lr=get_lr(optimizer))
    return
loss_meter


def valid_epoch(model, valid_loader):
    loss_meter = AvgMeter()

    tqdm_object
= tqdm(valid_loader, total=len(valid_loader))
    for batch in tqdm_object:
        batch = {k:
v.to(CFG.device) for k, v in batch.items() if k != "caption"}
        loss =
model(batch)

        count = batch["image"].size(0)

loss_meter.update(loss.item(), count)


tqdm_object.set_postfix(valid_loss=loss_meter.avg)
    return loss_meter


def main():

train_df, valid_df = make_train_valid_dfs()
    tokenizer =
DistilBertTokenizer.from_pretrained(CFG.text_tokenizer)
    train_loader =
build_loaders(train_df, tokenizer, mode="train")
    valid_loader =
build_loaders(valid_df, tokenizer, mode="valid")


    model =
CLIPModel().to(CFG.device)
    params = [
        {"params":
model.image_encoder.parameters(), "lr": CFG.image_encoder_lr},

{"params": model.text_encoder.parameters(), "lr": CFG.text_encoder_lr},

    {"params": itertools.chain(
            model.image_projection.parameters(),
model.text_projection.parameters()
        ), "lr": CFG.head_lr,
"weight_decay": CFG.weight_decay}
    ]
    optimizer = torch.optim.AdamW(params,
weight_decay=0.)
    lr_scheduler = torch.optim.lr_scheduler.ReduceLROnPlateau(

optimizer, mode="min", patience=CFG.patience, factor=CFG.factor
    )
    step =
"epoch"

    best_loss = float('inf')
    for epoch in range(CFG.epochs):

print(f"Epoch: {epoch + 1}")
        model.train()
        train_loss =
train_epoch(model, train_loader, optimizer, lr_scheduler, step)
```

```
        model.eval()

with torch.no_grad():
            valid_loss = valid_epoch(model, valid_loader)


 if valid_loss.avg < best_loss:
            best_loss = valid_loss.avg

torch.save(model.state_dict(), "best.pt")
            print("Saved Best
Model!")

        lr_scheduler.step(valid_loss.avg)



**********
**CLIP modifikuotas**
**********

import numpy as np
import
pandas as pd
import itertools
from tqdm.autonotebook import tqdm
import matplotlib.pyplot as
plt

import torch
from torch import nn
import torch.nn.functional as F
from transformers import
AutoModel, AutoTokenizer, AutoConfig

TSCOLS = 256
TRAIN_DATA =
"train_val_data/BTC_train.csv"

class CFG:
    debug = False
    batch_size = 32

num_workers = 0
    head_lr = 1e-3
    timeseries_encoder_lr = 1e-4
    text_encoder_lr = 1e-5

   weight_decay = 1e-3
    patience = 1
    factor = 0.8
    epochs = 20
    device =
torch.device("cuda" if torch.cuda.is_available() else "cpu")


model_name = 'resnet50'
    text_encoder_model = "distilbert-base-uncased"

text_embedding = 768
    text_tokenizer = "distilbert-base-uncased"

timeseries_embedding = 256
    max_length = 200

    pretrained = True # for  text encoder

trainable = True # for text encoder
    temperature = 1.0

    # image size
    # size = 224
```



```
  # for projection head; used for both image and text encoders
```

```
    num_projection_layers = 1

 projection_dim = 256
    dropout = 0.1

class AvgMeter:
    def __init__(self,
name="Metric"):
        self.name = name
        self.reset()

    def reset(self):

      self.avg, self.sum, self.count = [0] * 3

    def update(self, val, count=1):

self.count += count
        self.sum += val * count
        self.avg = self.sum / self.count


  def __repr__(self):
        text = f"{self.name}: {self.avg:.4f}"
        return
text

def get_lr(optimizer):
    for param_group in optimizer.param_groups:
        return
param_group["lr"]


class CLIPDataset(torch.utils.data.Dataset):
    def
__init__(self, timeseries, captions, tokenizer):
        self.timeseries = list(timeseries)

    self.captions = list(captions)
        self.encoded_captions = tokenizer(

list(captions), padding=True, truncation=True, max_length=CFG.max_length
        )

self.normalizer = 'none'

    def __getitem__(self, idx):
        item = {
            key:
torch.tensor(values[idx])
            for key, values in self.encoded_captions.items()

}

        if self.normalizer =='none':
            item['timeseries'] =
torch.tensor(self.timeseries[idx]).float()
        if self.normalizer =='minmax':

item['timeseries'] = (torch.tensor(self.timeseries[idx]).float() -
torch.tensor(self.timeseries[idx]).float().min()) /
(torch.tensor(self.timeseries[idx]).float().max() -
torch.tensor(self.timeseries[idx]).float().min())
        if self.normalizer =='meansd':

     item['timeseries'] = (torch.tensor(self.timeseries[idx]).float() -
torch.tensor(self.timeseries[idx]).float().mean()) /
torch.tensor(self.timeseries[idx]).float().std()

        item['caption'] =
self.captions[idx]

        return item
```

```
    def __len__(self):
        return
len(self.captions)

class TextEncoder(nn.Module):
    def __init__(self,
model_name=CFG.text_encoder_model, pretrained=CFG.pretrained, trainable=CFG.trainable):

super().__init__()
        if pretrained:
            self.model =
AutoModel.from_pretrained(model_name)
        else:
            self.model =
AutoModel(config=AutoConfig())

        for p in self.model.parameters():

  p.requires_grad = False

        # we are using the CLS token hidden representation as the
sentence's embedding
        self.target_token_idx = 0

    def forward(self, input_ids,
attention_mask):
        output = self.model(input_ids=input_ids,
attention_mask=attention_mask)
        last_hidden_state = output.last_hidden_state

return last_hidden_state[:, self.target_token_idx, :]

class ProjectionHead(nn.Module):

def __init__(
        self,
        embedding_dim,
        projection_dim=CFG.projection_dim,

      dropout=CFG.dropout
    ):
        super().__init__()
        self.projection =
nn.Linear(embedding_dim, projection_dim)
        self.gelu = nn.GELU()
        self.fc =
nn.Linear(projection_dim, projection_dim)
        self.dropout = nn.Dropout(dropout)

self.layer_norm = nn.LayerNorm(projection_dim)

    def forward(self, x):
        projected
= self.projection(x)
        x = self.gelu(projected)
        x = self.fc(x)
        x =
self.dropout(x)
        x = x + projected
        x = self.layer_norm(x)
        return x


class CLIPModel(nn.Module):
    def __init__(
        self,

temperature=CFG.temperature,
        text_embedding=CFG.text_embedding,

timeseries_embedding=CFG.timeseries_embedding,
        use_timeseries_projection_head=True,

):
        super().__init__()
```

```
        self.text_encoder = TextEncoder()

self.text_projection = ProjectionHead(embedding_dim=text_embedding)
        self.ts_projection
= ProjectionHead(embedding_dim=timeseries_embedding)
        self.temperature = temperature

    self.use_timeseries_projection_head = use_timeseries_projection_head

    def forward(self,
batch):
        timeseries_features = batch["timeseries"]
        text_features =
self.text_encoder(
            input_ids=batch["input_ids"],
attention_mask=batch["attention_mask"]
        )

        if
self.use_timeseries_projection_head:
            timeseries_embeddings =
self.ts_projection(timeseries_features)
        else:
            timeseries_embeddings =
timeseries_features

        text_embeddings =
self.text_projection(text_features)

        # Calculating the Loss
        logits =
(text_embeddings @ timeseries_embeddings.T) / self.temperature
        timeseries_similarity =
timeseries_embeddings @ timeseries_embeddings.T
        texts_similarity = text_embeddings @
text_embeddings.T
        targets = F.softmax(
            (timeseries_similarity +
texts_similarity) / 2 * self.temperature, dim=-1
        )
        texts_loss =
cross_entropy(logits, targets, reduction='none')
        images_loss = cross_entropy(logits.T,
targets.T, reduction='none')
        loss =  (images_loss + texts_loss) / 2.0 # shape:
(batch_size)
        return loss.mean()


def cross_entropy(preds, targets, reduction='none'):

   log_softmax = nn.LogSoftmax(dim=-1)
    loss = (-targets * log_softmax(preds)).sum(1)
    if
reduction == "none":
        return loss
    elif reduction == "mean":

   return loss.mean()

def make_train_valid_dfs(TRAIN_DATA_PATH):
    dataframe =
pd.read_csv(TRAIN_DATA_PATH, lineterminator='\n')
    max_id = dataframe.shape[0]

timeseries_ids = np.arange(0, max_id)
    np.random.seed(42)
    valid_ids = np.random.choice(

       timeseries_ids, size=int(0.2 * len(timeseries_ids)), replace=False
    )
    train_ids =
[id_ for id_ in timeseries_ids if id_ not in valid_ids]
    train_dataframe =
dataframe[dataframe.index.isin(train_ids)].reset_index(drop=True)
```

```
    valid_dataframe =
dataframe[dataframe.index.isin(valid_ids)].reset_index(drop=True)
    return train_dataframe,
valid_dataframe


def build_loaders(dataframe, tokenizer, mode):
    dataset = CLIPDataset(

    dataframe.iloc[:, -TSCOLS:].values,
        dataframe.iloc[:, 0].values,

tokenizer=tokenizer,
    )
    dataloader = torch.utils.data.DataLoader(
        dataset,

  batch_size=CFG.batch_size,
        num_workers=CFG.num_workers,
        shuffle=True if mode
== "train" else False,
    )
    return dataloader


def train_epoch(model,
train_loader, optimizer, lr_scheduler, step):
    loss_meter = AvgMeter()
    tqdm_object =
tqdm(train_loader, total=len(train_loader))
    for batch in tqdm_object:
        batch = {k:
v.to(CFG.device) for k, v in batch.items() if k != "caption"}
        loss =
model(batch)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

    if step == "batch":
            lr_scheduler.step()

        count =
batch["timeseries"].size(0)
        loss_meter.update(loss.item(), count)


tqdm_object.set_postfix(train_loss=loss_meter.avg, lr=get_lr(optimizer))
    return
loss_meter


def valid_epoch(model, valid_loader):
    loss_meter = AvgMeter()

    tqdm_object
= tqdm(valid_loader, total=len(valid_loader))
    for batch in tqdm_object:
        batch = {k:
v.to(CFG.device) for k, v in batch.items() if k != "caption"}
        loss =
model(batch)

        count = batch["timeseries"].size(0)

loss_meter.update(loss.item(), count)


tqdm_object.set_postfix(valid_loss=loss_meter.avg)
    return loss_meter

def
get_ts_embeddings(valid_df, model_path):
    tokenizer =
AutoTokenizer.from_pretrained(CFG.text_tokenizer)
```

```
    valid_loader = build_loaders(valid_df,
tokenizer, mode="valid")

    model = CLIPModel()

model.load_state_dict(torch.load(model_path) if CFG.device == "cuda" else
torch.load(model_path, map_location=torch.device('cpu')))
    model.eval()


valid_timeseries_embeddings = []
    with torch.no_grad():
        for batch in
tqdm(valid_loader):

valid_timeseries_embeddings.append(batch["timeseries"])
    return model,
torch.cat(valid_timeseries_embeddings)

def find_matches(model, timeseries_embeddings, query,
n=9):
    tokenizer = AutoTokenizer.from_pretrained(CFG.text_tokenizer)
    encoded_query =
tokenizer([query])
    batch = {
        key: torch.tensor(values)
        for key, values in
encoded_query.items()
    }
    with torch.no_grad():
        text_features =
model.text_encoder(
            input_ids=batch["input_ids"],
attention_mask=batch["attention_mask"]
        )
        text_embeddings =
model.text_projection(text_features)

    timeseries_embeddings_n =
F.normalize(timeseries_embeddings, p=2, dim=-1)
    text_embeddings_n =
F.normalize(text_embeddings, p=2, dim=-1)
    dot_similarity = text_embeddings_n @
timeseries_embeddings_n.T

    values, indices = torch.topk(dot_similarity.squeeze(0), n *
5)
    matches =  indices[::5]

    return matches

def main(TRAIN_DATA_PATH, WEIGHTS_PATH):

 train_df, valid_df = make_train_valid_dfs(TRAIN_DATA_PATH)
    tokenizer =
AutoTokenizer.from_pretrained(CFG.text_tokenizer)
    train_loader = build_loaders(train_df,
tokenizer, mode="train")
    valid_loader = build_loaders(valid_df, tokenizer,
mode="valid")


    model = CLIPModel().to(CFG.device)
    params = [

{"params": {}, "lr": CFG.timeseries_encoder_lr},

{"params": model.text_encoder.parameters(), "lr": CFG.text_encoder_lr},

    {"params": itertools.chain({}, model.text_projection.parameters()
        ),
"lr": CFG.head_lr, "weight_decay": CFG.weight_decay}
    ]
    optimizer =
torch.optim.AdamW(params, weight_decay=0.)
```

```
    lr_scheduler =
torch.optim.lr_scheduler.ReduceLROnPlateau(
        optimizer, mode="min",
patience=CFG.patience, factor=CFG.factor
    )
    step = "epoch"

    best_loss =
float('inf')
    for epoch in range(CFG.epochs):
        print(f"Epoch: {epoch +
1}")
        model.train()
        train_loss = train_epoch(model, train_loader,
optimizer, lr_scheduler, step)
        model.eval()
        with torch.no_grad():

valid_loss = valid_epoch(model, valid_loader)

        if valid_loss.avg <
best_loss:
            best_loss = valid_loss.avg
            torch.save(model.state_dict(),
WEIGHTS_PATH)
            print("Saved Best Model!")


lr_scheduler.step(valid_loss.avg)


**********************
**Duomenu
griauzimas**
**********************

import twint
import pandas as pd
import datetime
import
sys

def count_lines(filename):
    with open(filename) as fp:
        count = 0
        for _
in fp:
            count += 1
    return count

TICKER = sys.argv[1]

c =
twint.Config()
c.Search = '$' + TICKER
c.Hide_output = True
c.Store_csv = True
c.Custom_csv =
["language", "created_at", "tweet",
"username"]

start_date = datetime.date(int(sys.argv[2]), int(sys.argv[3]),
int(sys.argv[4]))
end_date   = datetime.date(2023, 2, 19)

date_generated = [ str(start_date +
datetime.timedelta(n)) for n in range(int ((end_date - start_date).days))]

for i in
range(len(date_generated) - 1):
    c.Since = date_generated[i]
    c.Until =
date_generated[i+1]
    filename = "scraped_data/" + TICKER + date_generated[i] +
".csv"
```

```
    c.Output =  filename
    twint.run.Search(c)
    tlist =
c.search_tweet_list
    print(len(tlist))
    print(date_generated[i], count_lines(filename))


************************
**Parametru vertinimas**
************************

from
CLIP_timeseries import main
import sys

main(sys.argv[1],
sys.argv[2])

************************
**HPC paleidimo
kodas**
************************

#!/bin/sh
#SBATCH -p gpu
#SBATCH -n1
#SBATCH --gres
gpu
#SBATCH -t 300
. gpu_env/bin/activate
python3 train.py $1
$2


***********************************
**Kriptovaliutu duomenu
paruosimas**
***********************************

#!/usr/bin/env python
# coding:
utf-8

import pandas as pd

IN_DATA_FOLDER = "raw_data/prices/"
OUT_DATA_FOLDER =
"clean_data/prices/"

def read_from_cryptodatadownload(filename):
    # This
particular site returns data with 1 row of their link to website
    # Also the time series is
in reverse so need to do adjustments
    cols = ["Date", "Open",
"High", "Low", "Close", "Volume USDT"]
    df =
pd.read_csv(filename, skiprows = 1, parse_dates = ['Date'])[::-1].reset_index(drop = True)

df = df[cols]
    df = df.rename({'Volume USDT' : 'Volume'}, axis = 1)
    return df

# ##
Day

read_from_cryptodatadownload(IN_DATA_FOLDER +
"Binance_BTCUSDT_d.csv").to_csv(OUT_DATA_FOLDER + "BTCUSDT_day.csv", index
= False)
read_from_cryptodatadownload(IN_DATA_FOLDER +
"Binance_ETHUSDT_d.csv").to_csv(OUT_DATA_FOLDER + "ETHUSDT_day.csv", index
= False)
read_from_cryptodatadownload(IN_DATA_FOLDER +
"Binance_XRPUSDT_d.csv").to_csv(OUT_DATA_FOLDER + "XRPUSDT_day.csv", index
```

```
= False)

# ## Hour

read_from_cryptodatadownload(IN_DATA_FOLDER +
"Binance_BTCUSDT_1h.csv").to_csv(OUT_DATA_FOLDER + "BTCUSDT_hour.csv",
index = False)
read_from_cryptodatadownload(IN_DATA_FOLDER +
"Binance_ETHUSDT_1h.csv").to_csv(OUT_DATA_FOLDER + "ETHUSDT_hour.csv",
index = False)
read_from_cryptodatadownload(IN_DATA_FOLDER +
"Binance_XRPUSDT_1h.csv").to_csv(OUT_DATA_FOLDER + "XRPUSDT_hour.csv",
index = False)

# ## Minute

def read_minute_from_cryptodatadownload(ticker, year):
    # This
particular site returns data with 1 row of their link to website
    # Also the time series is
in reverse so need to do adjustments

    filename = "Binance_" + ticker +
"USDT_" + year + "_minute.csv"
    cols = ['date', 'open', 'high', 'low',
'close', "Volume USDT"]
    df = pd.read_csv(IN_DATA_FOLDER + filename, skiprows = 1,
parse_dates = ['date'])[::-1].reset_index(drop = True)
    df = df[cols]
    df = df.rename({

      "Volume USDT" : 'Volume',
        'date' : 'Date',
        'open' : 'Open',

    'high' : 'High',
        'low' : 'Low',
        'close' : 'Close',
    }, axis = 1)

return df

for ticker in ["BTC", "ETH", "XRP"]:
    df =
read_minute_from_cryptodatadownload(ticker, "2021")
    df.to_csv(OUT_DATA_FOLDER +
ticker + "USDT_minute.csv", index = False)


*****************************
**Twitter duomenu
paruosimas**
*****************************

#!/usr/bin/env python
# coding: utf-8

import
pandas as pd
from tqdm import tqdm
import os

IN_DATA_FOLDER =
'raw_data/tweets/'
OUT_DATA_FOLDER = 'clean_data/tweets/'
TICKERS = ["BTC",
"ETH", "XRP"]

def clean_df(data):
    # My collected data about tweets
contains a lot of unusable information
    # So removing that plus some sanity check filtering

   df = data.copy()
    df = df[df.language == 'en']
```

```
    df["timestamp"] =
pd.to_datetime(df.date + " " + df.time)
    df = df[['tweet', 'username',
'timestamp']]
    df = df.dropna()
    df = df.reset_index(drop=True)
    return df


for
ticker in TICKERS:
    files = [f for f in os.listdir(IN_DATA_FOLDER) if f.startswith(ticker)]

   dfs = []
    for f in tqdm(files):
        temp = pd.read_csv(IN_DATA_FOLDER + f,
engine='python')[['language', 'date', 'time','tweet', 'username']]
        dfs.append(temp)

df = pd.concat(dfs)
    df = clean_df(df)
    df.to_csv(OUT_DATA_FOLDER + ticker +
"_tweets.csv", index=False)

**********
**Paveiksleliu
kurimas**
**********

#!/usr/bin/env python
# coding: utf-8

import pandas as
pd
import librosa
import librosa.display
import numpy as np
import matplotlib.pyplot as
plt
from tqdm import tqdm

BTC_OUT_DATA_FOLDER =
"librosa-images/"
ETH_OUT_DATA_FOLDER =
"eth-librosa-images/"
XRP_OUT_DATA_FOLDER = "xrp-librosa-images/"

def
visualize_series(timeseries, image_name, out_folder):
    chroma =
librosa.feature.chroma_stft(S=np.abs(librosa.stft(timeseries, n_fft=256)), sr=4000)
    fig =
plt.figure(frameon=False)
    fig.set_size_inches(1, 1)
    img =
librosa.display.specshow(chroma)
    fig.savefig(out_folder + image_name + '.png',
bbox_inches='tight', pad_inches=0)
    plt.close()

btc =
pd.read_csv("train_val_data/BTC_train.csv", lineterminator='\n')

for i in
tqdm(range(min(10000, btc.shape[0]))):
    timeseries = np.array(btc.iloc[i, 1:],
dtype='float32')
    visualize_series(timeseries, str(i), BTC_OUT_DATA_FOLDER)

eth =
pd.read_csv("train_val_data/ETH_train.csv", lineterminator='\n')

for i in
tqdm(range(min(10000, eth.shape[0]))):
    timeseries = np.array(eth.iloc[i, 1:],
```

```
dtype='float32')
    visualize_series(timeseries, str(i), ETH_OUT_DATA_FOLDER)

xrp =
pd.read_csv("train_val_data/XRP_train.csv", lineterminator='\n')

for i in
tqdm(range(min(10000, xrp.shape[0]))):
    timeseries = np.array(xrp.iloc[i, 1:],
dtype='float32')
    visualize_series(timeseries, str(i),
XRP_OUT_DATA_FOLDER)

***************************
**Duomenu aibiu
paruosimas**
***************************

#!/usr/bin/env python
# coding: utf-8

import pandas
as pd
from tqdm import tqdm
from sklearn.model_selection import
train_test_split

TIME_SERIES_SIZE = 256
IN_PRICES_DATA_FOLDER =
"clean_data/prices/"
IN_TWEETS_DATA_FOLDER =
"clean_data/tweets/"
OUT_DATA_FOLDER = "train_val_data/"
USEROWS =
100000

def prepare_dataset(ticker):
    tweets = pd.read_csv(IN_TWEETS_DATA_FOLDER + ticker +
"_tweets.csv", lineterminator='\n', parse_dates=['timestamp']).head(USEROWS)

prices = pd.read_csv(IN_PRICES_DATA_FOLDER + ticker + "USDT_minute.csv",
parse_dates=['Date'])
    prices["price"] = (prices.Open + prices.Close) / 2

timeseries = pd.DataFrame([], columns=[str(i) for i in range(TIME_SERIES_SIZE)])
    indexes =
[]
    for i in tqdm(range(tweets.timestamp.shape[0])):
        tempdata =
prices[prices.Date>=tweets.timestamp[i]].price.head(TIME_SERIES_SIZE)
        tempdata =
(100 * (tempdata / tempdata.iat[0] - 1))
        if tempdata.shape[0] != 0:

indexes.append(i)
            tempdf = pd.DataFrame(tempdata.array.reshape(1,
TIME_SERIES_SIZE), columns=[str(i) for i in range(TIME_SERIES_SIZE)])
            timeseries =
pd.concat([timeseries, tempdf])
    timeseries.reset_index(drop=True, inplace=True)
    df =
pd.concat([tweets.iloc[indexes], timeseries], axis=1)
    df = df.drop(['username',
'timestamp'], axis=1)
    train, val = train_test_split(df, test_size=0.25, random_state=42)

 train.to_csv("train_val_data/" + ticker + "_train.csv", index=False)

val.to_csv("train_val_data/" + ticker + "_val.csv", index=False)

for
ticker in ["BTC", "ETH", "XRP"]:

prepare_dataset(ticker)
```

```
*********************************
**Parametru vertinimo
paleidimas**
*********************************

#!/usr/bin/env python
# coding: utf-8

from
CLIP import main

main()

******************
**Darbo
grafikai**
******************

#!/usr/bin/env python
# coding: utf-8

import pandas as pd
from
tqdm import tqdm
import matplotlib.pyplot as plt
plt.rcParams.update({'font.size': 15})

ls
clean_data/prices

ls clean_data/tweets

# # Tweet count x price plots

# ## BTC

p =
pd.read_csv("clean_data/prices/BTCUSDT_day.csv", parse_dates=['Date'])[['Date',
'Close', 'Volume']]
p.head()

t = pd.read_csv("clean_data/tweets/BTC_tweets.csv",
lineterminator='\n', parse_dates = ['timestamp'])
t['Date'] = t.timestamp.dt.date
t['counter']
= 1
t.head()

tt = t.groupby(['Date']).counter.value_counts()
tt = tt.reset_index(name =
'total')[['Date', 'total']]
tt.Date = pd.to_datetime(tt.Date)
tt.head()

j = pd.merge(p,tt,
on='Date')
j.head()

j.corr()

fig, ax1 = plt.subplots(figsize = (15, 7))

ax2 =
ax1.twinx()
ax2.plot(j.Date, j.total, 'b-', alpha=0.8)
ax1.plot(j.Date, j.Close, 'g-',
alpha=0.8)

ax1.set_xlabel('Data', fontsize=20)
ax1.set_ylabel('Kaina', color='g',
fontsize=20)
ax2.set_ylabel('?raš? kiekis', color='b',
fontsize=20)
```

```
plt.savefig('btc_close_count.png', dpi=100)
plt.show()

fig, ax1 =
plt.subplots(figsize = (15, 7))

ax2 = ax1.twinx()
ax2.plot(j.Date, j.total, 'b-',
alpha=0.8)
ax1.plot(j.Date, j.Volume, 'g-', alpha=0.8)

ax1.set_xlabel('Data',
fontsize=20)
ax1.set_ylabel('Suprekiautas kiekis', color='g',
fontsize=20)
ax2.set_ylabel('?raš? kiekis', color='b',
fontsize=20)

plt.savefig('btc_volume_count.png', dpi=100)
plt.show()


# ## XRP

p =
pd.read_csv("clean_data/prices/XRPUSDT_day.csv", parse_dates=['Date'])[['Date',
'Close', 'Volume']]
p.head()

t = pd.read_csv("clean_data/tweets/XRP_tweets.csv",
lineterminator='\n', parse_dates = ['timestamp'])
t['Date'] = t.timestamp.dt.date
t['counter']
= 1
t.head()

tt = t.groupby(['Date']).counter.value_counts()
tt = tt.reset_index(name =
'total')[['Date', 'total']]
tt.Date = pd.to_datetime(tt.Date)
tt.head()

j = pd.merge(p,tt,
on='Date')
j.head()

j.corr()

fig, ax1 = plt.subplots(figsize = (15, 7))

ax2 =
ax1.twinx()
ax2.plot(j.Date, j.total, 'b-', alpha=0.8)
ax1.plot(j.Date, j.Close, 'y-',
alpha=0.8)

ax1.set_xlabel('Data', fontsize=20)
ax1.set_ylabel('Kaina', color='y',
fontsize=20)
ax2.set_ylabel('?raš? kiekis', color='b',
fontsize=20)

plt.savefig('xrp_close_count.png', dpi=100)
plt.show()

fig, ax1 =
plt.subplots(figsize = (15, 7))

ax2 = ax1.twinx()
ax2.plot(j.Date, j.total, 'b-',
alpha=0.8)
ax1.plot(j.Date, j.Volume, 'y-', alpha=0.8)

ax1.set_xlabel('Data',
fontsize=20)
```

```
ax1.set_ylabel('Suprekiautas kiekis', color='b',
fontsize=20)
ax2.set_ylabel('?raš? kiekis', color='y',
fontsize=20)

plt.savefig('xrp_volume_count.png', dpi=100)
plt.show()


# # Price change
plots

plt.rcParams.update({'font.size': 22})


# ## BTC

btc =
pd.read_csv("train_val_data/BTC_train.csv",
lineterminator='\n')
plt.figure(figsize=(15,
7))
plt.xlabel("Indeksas")
plt.ylabel("Kainos pokytis %")
plt.plot([i for i
in range(256)], btc.iloc[0, 1:])
plt.savefig('btc_pct_change.png')


# ## ETH

eth =
pd.read_csv("train_val_data/ETH_train.csv",
lineterminator='\n')
plt.figure(figsize=(15,
7))
plt.xlabel("Indeksas")
plt.ylabel("Kainos pokytis %")
plt.plot([i for i
in range(256)], eth.iloc[0, 1:])
plt.savefig('eth_pct_change.png')


# ## XRP

xrp =
pd.read_csv("train_val_data/XRP_train.csv",
lineterminator='\n')
plt.figure(figsize=(15,
7))
plt.xlabel("Indeksas")
plt.ylabel("Kainos pokytis %")
plt.plot([i for i
in range(256)], xrp.iloc[0, 1:])
plt.savefig('xrp_pct_change.png')


# # Training and
validation loss plots

# ## Original CLIP

def get_losses(data):
    tl = []
    vl = []


for i in df:
        if 'train_loss' in i:

tl.append(float(i.split("train_loss=")[1].split("]")[0]))

        if
'valid_loss' in i:
```

```
vl.append(float(i.split("valid_loss=")[1].split("]")[0]))

print("Lengths", len(tl), len(vl))
    return tl, vl


# ### BTC

df =
pd.read_fwf('training_logs/BTC_images_100_epoch_10000.txt').INTRO.values
btc_tl, btc_vl =
get_losses(df)
n = 50
ids = [i for i in range(n)]

plt.figure(figsize=(8, 6),
dpi=100)
plt.xlabel("Epocha")
plt.plot(ids, btc_tl[:n], label = 'Mokymosi
paklaida')
plt.plot(ids, btc_vl[:n], label = 'Validavimo paklaida')
plt.legend(loc="upper
right")
plt.savefig('image_loss.png', dpi=100)


# ### ETH

df =
pd.read_fwf('training_logs/ETH_images_50_epoch_10000.txt').INTRO.values
eth_tl, eth_vl =
get_losses(df)
n = len(eth_tl)
ids = [i for i in range(n)]
plt.figure(figsize=(8, 6),
dpi=100)
plt.xlabel("Epocha")
plt.plot(ids, eth_tl[:n], label = 'Mokymosi
paklaida')
plt.plot(ids, eth_vl[:n], label = 'Validavimo paklaida')
plt.legend(loc="upper
right")
plt.savefig('eth_mage_loss.png', dpi=100)


# ### XRP

df =
pd.read_fwf('training_logs/XRP_images_50_epoch_10000.txt').INTRO.values
eth_tl, eth_vl =
get_losses(df)
n = len(eth_tl)
ids = [i for i in range(n)]
plt.figure(figsize=(8, 6),
dpi=100)
plt.xlabel("Epocha")
plt.plot(ids, eth_tl[:n], label = 'Mokymosi
paklaida')
plt.plot(ids, eth_vl[:n], label = 'Validavimo paklaida')
plt.legend(loc="upper
right")
plt.savefig('xrp_mage_loss.png', dpi=100)


# ## Modified CLIP

def
parse_output(filename, imagepath):
    file = open(filename,mode='r')
    df = file.read()

file.close()

    trains = []
    valids = []
```

```
    byepoch = df.split('Epoch')
    for
epoch in tqdm(byepoch[1:]):
        a = []
        b = []
        byline = epoch.split('\n')

     for i in byline:
            if 'train_loss' in i:

a.append(float(i.split("train_loss=")[1].split("]")[0]))
            if
'valid_loss' in i:

b.append(float(i.split("valid_loss=")[1].split("]")[0]))

trains.append(a[-1])
        valids.append(b[-1])

    n = len(trains)
    ids = [i for
i in range(n)]
    plt.figure(figsize=(8, 6), dpi=100)
    plt.xlabel("Epocha")

plt.plot(ids, trains[:n], label = 'Mokymosi paklaida')
    plt.plot(ids, valids[:n], label =
'Validavimo paklaida')
    plt.legend(loc="upper right")
    plt.savefig(imagepath,
dpi=100)


parse_output('training_logs/BTC_100000_training.txt',
'btc_ts_loss.png')

parse_output('training_logs/ETH_100000_training.txt',
'eth_ts_loss.png')

parse_output('training_logs/XRP_100000_training.txt',
'xrp_ts_loss.png')

************************
**Paveiksliuku
spejimai**
************************

#!/usr/bin/env python
# coding: utf-8

# In[87]:


# !pip
install transformers
import librosa
import librosa.display
import pandas as pd
import numpy as
np
from tqdm.autonotebook import tqdm
import matplotlib.pyplot as plt

from CLIPIMAGES import
get_image_embeddings, find_image_matches, make_train_valid_dfs

def
visualize_series(timeseries, image_name):
    chroma =
librosa.feature.chroma_stft(S=np.abs(librosa.stft(timeseries, n_fft=256)), sr=4000)
    fig =
plt.figure(frameon=False)
    fig.set_size_inches(1, 1)
    img =
```

```
librosa.display.specshow(chroma)
    fig.savefig(image_name + '.png', bbox_inches='tight',
pad_inches=0)
    plt.close()

def plot_matches(values):
    emptycount = 0

plt.figure(figsize=(20,10))
    for i in values:
        if len(np.unique(np.array(i))) == 1:

          emptycount+=1
        else:
            plt.plot(i)

    print("Out of
", len(values), " matches, ", emptycount, " are empty")



train_df, valid_df = make_train_valid_dfs()
model, image_embeddings =
get_image_embeddings(train_df, "/content/drive/MyDrive/BACHELOR'S
DATA/weights/BTC_IMAGES_10000.pt")

matches = find_image_matches(model,

image_embeddings,
             query="The steady rise in BTC price not only reflects its
immense value as a digital asset, but also offers a promising outlook for investors, fueling
excitement and confidence in its future trajectory.",

image_filenames=train_df['image'].values,
             n=9)

matches

p =
df.iloc[[int(i.split('.')[0]) for i in matches], :].reset_index(drop = True)

toplot = []
for i
in range(len(p)):
  toplot.append(p.iloc[i,
1:].values)

plot_matches(toplot)
np.array(toplot).mean(axis =
0).mean()

plt.figure(figsize=(10,7))

plt.plot(np.array(toplot).mean(axis=0))
plt.savefig(&quo
t;btc_images_pos_pred_mean.png")
np.array(toplot).mean(axis=0).mean()

matches =
find_image_matches(model,
             image_embeddings,
             query="The volatile
nature of BTC price can make it challenging for investors to predict and navigate, causing
anxiety and uncertainty in the market.",

image_filenames=valid_df['image'].values,
             n=9)

matches

p =
df.iloc[[int(i.split('.')[0]) for i in matches], :].reset_index(drop = True)

toplot = []
for i
```

```
in range(len(p)):
  toplot.append(p.iloc[i,
1:].values)

plot_matches(toplot)
np.array(toplot).mean(axis =
0).mean()

plt.figure(figsize=(10,7))

plt.plot(np.array(toplot).mean(axis=0))
plt.savefig(&quo
t;btc_images_neg_pred_mean.png")
np.array(toplot).mean(axis=0).mean()

******************
*******
**Laiko eiluciu spejimai**
*************************


#!/usr/bin/env python
# coding:
utf-8

import pandas as pd
import numpy as np
from tqdm.autonotebook import tqdm
import
matplotlib.pyplot as plt
plt.rcParams.update({'font.size': 20})

from scripts.CLIPTS import
get_ts_embeddings, find_ts_matches

def plot_matches(values):
    emptycount = 0

plt.figure(figsize=(20,10))
    for i in values:
        if len(np.unique(np.array(i))) == 1:

          emptycount+=1
        else:
            plt.plot(i)

    print("Out of
", len(values), " matches, ", emptycount, " are empty")

def
read(path):
    return pd.read_csv(path, lineterminator='\n')

def
get_model_embeddings(data_path, model_path):
    df = read(data_path)
    return
get_ts_embeddings(df, model_path)


# # Modified CLIP

# ## BTC

model, ts_embeddings =
get_model_embeddings("train_val_data/BTC_val.csv",
"weights/BTC_100000.pt")


matches = find_ts_matches(model,

ts_embeddings,
             query="The steady rise in BTC price not only reflects its
immense value as a digital asset, but also offers a promising outlook for investors, fueling
excitement and confidence in its future trajectory.",
```

```
n=10)
plot_matches(ts_embeddings[matches])
plt.savefig("btc_ts_pred.png")

plt.figure
(figsize=(10,7))
plt.plot(ts_embeddings[matches].mean(axis=0))
plt.savefig("btc_ts_pred_me
an.png")
ts_embeddings[matches].mean(axis=0).mean()

matches = find_ts_matches(model,

          ts_embeddings,
              query="The volatile nature of BTC price can make it
challenging for investors to predict and navigate, causing anxiety and uncertainty in the
market.",

n=10)
plot_matches(ts_embeddings[matches])
plt.savefig("btc_ts_neg_pred.png")

plt.fi
gure(figsize=(10,7))

plt.plot(ts_embeddings[matches].mean(axis=0))
plt.savefig("btc_ts_ne
g_pred_mean.png")
ts_embeddings[matches].mean(axis=0).mean()


import os

pos =
['860.png',
 '2247.png',
 '333.png',
 '274.png',
 '988.png',
 '1148.png',
 '1080.png',

'779.png',
 '1359.png',
 '1895.png']

neg = ['3350.png',
 '8990.png',
 '1448.png',

'1259.png',
 '3928.png',
 '4687.png',
 '4374.png',
 '2999.png',
 '5583.png',
 '7758.png']


for
i in pos:
    print(i)
    os.system("cp
timeseries_images/btc-librosa-images/"+i+" .")


for i in neg:
    print(i)

os.system("cp timeseries_images/btc-librosa-images/"+i+" .")
```