

STRESZCZENIE

Przewidywania giełdowe oparte o uczenie maszynowe są przedmiotem wielu badań. Niestrywialność tego problemu jest spowodowana dużą liczbą czynników. Najprostsze techniki oparte są na samych danych z giełdy. W naszym projekcie w przewidywaniu korzystamy z danych giełdy oraz sentymentu społecznego. Głównym celem zrealizowanego projektu było stworzenie systemu wspierającego proces decyzyjny obrotu akcjami, który wykorzystuje dane giełdowe i dane z platformy społecznościowej *Twitter*. Celem pośrednim było zbadanie skuteczności przewidywania przez sieć neuronową *LSTM* ceny wybranego aktywa finansowego na następny dzień w zależności od danych wejściowych. Rezultatem projektu jest parametryzowalna sieć neuronowa, silnik do przetwarzania danych z *Twittera* oraz przeglądarkowa aplikacja użytkownika, która pozwala na stworzenie zbioru treningowego, trening oraz test sieci.

W pracy przedstawiliśmy alternatywne podejścia do zagadnienia przewidywania, architekturę stworzonego systemu, sposób przygotowania danych, opis i działanie modelu sieci neuronowej *LSTM*, działanie naszej aplikacji użytkownika wraz z wynikami oraz dalsze możliwe pola rozwoju projektu.

Słowa kluczowe: sieć neuronowa *LSTM*, sentyment, platforma *Twitter*, giełda

ABSTRACT

Stock market prediction based on machine learning is the subject of much research. The non-triviality of this problem is due to a large number of factors. The simplest techniques are based on the stock market data itself. In our project, we use stock market data and public sentiment in prediction. The main goal of the completed project was to create a system to support the decision-making process of stock trading, which uses stock market data and data from the social platform Twitter. The intermediate goal was to test the effectiveness of the neural network LSTM in predicting the price of a selected financial asset for the next day depending on the input data. The result of the project is a parameterizable neural network, an engine for processing data from Twitter and a browser user application, that allows to create a training set, train and test the network.

In this paper, we presented alternative approaches to the prediction problem, the architecture of the created system, how we prepare the data, the description and how the neural network model LSTM works, the operation of our user application with the results, and further possible improvements for the project.

Keywords: neural network LSTM, sentiment, Twitter, stock market

SPIS TREŚCI

Wykaz ważniejszych oznaczeń i skrótów.....	5
1. Wstęp (<i>Grzegorz Pozorski</i>)	6
1.1. Metody analiz rynku	6
1.2. Giełda	6
1.3. Definicja problemu	7
1.4. Cel pracy	7
1.5. Przegląd rozwiązań	7
1.5.1. Predicting Stock Movement Using Sentiment Analysis of Twitter Feed with Neural Networks	7
1.5.2. Stock Prediction Using Twitter Sentiment Analysis	8
1.5.3. A hybrid model integrating deep learning with investor sentiment analysis for stock price prediction	8
1.5.4. Using Neural Networks to Forecast Stock Market Prices	9
1.5.5. Stock Market Prediction Using Artificial Neural Networks	9
1.5.6. Using Deep Learning to Develop a Stock Price Prediction Model Based on Individual Investor Emotions	9
2. Architektura systemu (<i>Grzegorz Pozorski</i>)	10
2.1. Realizacja architektury	10
2.2. Wykorzystanie protokołu <i>HTTP</i>	11
2.3. Infrastruktura	11
3. Przygotowanie danych (<i>Jeremi Ledwoń, Grzegorz Pozorski</i>)	13
3.1. Przygotowanie danych z giełdy (<i>Jeremi Ledwoń</i>)	13
3.1.1. Format danych	13
3.1.2. Interwał	14
3.1.3. Sposób pozyskiwania danych	14
3.2. Przygotowanie danych z <i>Twittera</i> (<i>Grzegorz Pozorski</i>)	17
3.2.1. Pozyskanie tweetów	17
3.2.2. Dostępne zbiory danych	17
3.2.3. Filtrowanie tweetów	17
3.3. Analiza sentymentu (<i>Grzegorz Pozorski</i>)	18
3.3.1. Działanie biblioteki <i>VADER</i>	19
3.3.2. Działanie biblioteki <i>Flair</i>	19
3.4. Przetwarzanie pobranych tweetów (<i>Grzegorz Pozorski</i>)	19
3.5. Korekcja danych (<i>Jeremi Ledwoń</i>)	20
4. Model (<i>Jeremi Ledwoń</i>)	22
4.1. Budowa sieci neuronowej	22
4.1.1. Sieci rekurencyjne	22
4.1.2. LSTM	23
4.2. Trening sieci neuronowej	23
4.2.1. Faza treningu	24
4.2.2. Faza walidacji	24

4.2.3. Faza Testu.....	25
4.3. <i>PyTorch Lightning</i>	25
4.4. Konfiguracja sieci i znaczenie parametrów	26
4.4.1. Optymalizacja konfiguracji	26
4.5. Wstępne przetworzenie danych.....	27
4.5.1. Uzupełnienie brakujących cech	27
4.5.2. Podział zbioru na podzbiory	28
4.5.3. Normalizacja danych	28
4.6. Monitorowanie i analiza wyników.....	29
4.6.1. Wandb.....	29
4.6.2. Skuteczność.....	30
5. Warstwa logiki biznesowej (<i>Jeremi Ledwoń</i>)	31
5.1. Endpointy	31
5.1.1. /train.....	31
5.1.2. /predict	33
5.1.3. Endpointy informacyjne	33
6. Aplikacja użytkownika (<i>Grzegorz Pozorski</i>)	35
6.1. Wybrana technologia	35
6.1.1. <i>ReactJs</i>	35
6.1.2. Fragmenty kodu	36
6.1.3. Najważniejsze wykorzystane zależności.....	36
6.2. Przykład użycia.....	36
6.2.1. Generowanie sparametryzowanego modelu sieci neuronowej	36
6.2.2. Tworzenie zbioru danych z <i>Twittera</i>	38
6.2.3. Przewidywanie ceny danego aktywa finansowego na kolejny dzień	39
7. Podsumowanie (<i>Jeremi Ledwoń, Grzegorz Pozorski</i>).....	41
7.1. Wyniki sieci neuronowej (<i>Jeremi Ledwoń</i>)	41
7.2. Korelacja między sentymentem, a ceną aktywa finansowego (<i>Grzegorz Pozorski</i>)	42
7.2.1. Testy działania bibliotek, które liczą sentyment	42
7.2.2. Korelacja między sentymentem , a ceną danego aktywa.....	43
7.3. Możliwe usprawnienia (<i>Grzegorz Pozorski</i>)	44
7.3.1. Możliwe usprawnienia w zakresie sieci neuronowych.....	44
7.3.2. Możliwe usprawnienia w zakresie użytych danych	44
7.3.3. Możliwe usprawnienia w zakresie analizy sentymentu	45
7.4. Wnioski na temat wyboru architektury (<i>Grzegorz Pozorski</i>).....	46
Bibliografia	48
Dodatek A: Listingi	49

WYKAZ WAŻNIEJSZYCH OZNACZEŃ I SKRÓTÓW

API – Application Programming Interface

POMS – Profile of Mood States

DJIA – Dow Jones Industrial Average - wskaźnik kursów akcji 30 największych amerykańskich korporacji przemysłowych

SOFNN – Self Organizing Fuzzy Neural Networks

1. WSTĘP

Wraz ze wzrostem liczby informacji, generowanych przez ludzkość, nastąpił rozwój technologii informacyjnych. Przetwarzanie, analizowanie i wyciąganie wniosków na podstawie dużych zbiorów danych stanowi mniejszy lub większy obszar działalności każdej współczesnej korporacji. Dzięki rozbudowanym narzędziom takim jak np. języki programowania, *frameworki*, czy dostępne i darmowe *API*, każdy może zmierzyć się z dużymi zbiorami danych.

Ich przetwarzanie pomaga znajdować zależności, wzory, a w efekcie pozwala prognozować ich przyszłe zachowania.

Pomysł wykorzystania analizy danych giełdowych tak, by zarobić, zawsze rozpala umysły młodych studentów (tak było w naszym przypadku). Od kiedy pojawiła się giełda, ludzie szukają sposobu jak na niej zarobić. Przez lata powstawały rozmaite metody wnioskowania przyszłych cen akcji. Ich celem jest odpowiedź na proste pytania: 'kup?', 'sprzedaj?', 'czekaj?'.

1.1. Metody analiz rynku

Obecnie wśród najpopularniejszych metod możemy wyróżnić dwie: analizę techniczną oraz analizę fundamentalną. Pierwsza przewiduje zmiany kursów akcji na podstawie przeszłych wykresów cen akcji, wartości obrotów, wielkości zleceń oraz wskaźników technicznych. Druga rozpatruje stan gospodarki oraz emitenta akcji. Współcześnie stosowanie tych analiz w inwestowaniu daje niewiele lepsze rezultaty niż losowe podejmowanie decyzji. Z jakich więc narzędzi korzystać, by uzyskać lepszy wynik? Odpowiedź przychodzi na początek dwudziestego wieku.

Wtedy to powstaje analiza sentymentu, która ma zastosowanie w zupełnie innym zagadnieniu - pomaga zidentyfikować stany emocjonalne autora danej wypowiedzi. Przykładowo pozwala ocenić nastawienie konsumentów do marki.

Obecnie mamy możliwość analizowania sentymentu dużych grup ludzi. W jaki sposób? Możemy założyć, że treści publikowane w mediach społecznościowych przykładowo takich jak *Twitter*, *Facebook*, *Instagram*, czy *YouTube* odzwierciedlają emocje.

1.2. Giełda

Giełda to sesje handlowe realizowane w ustalonym, powszechnie znanym miejscu i czasie. W trakcie sesji brokerzy lub gracze giełdowi kupują i sprzedają papiery wartościowe, takie jak akcje i obligacje, po cenach ogłoszonych w codziennych notowaniach. Spółki pozyskują pieniądze na giełdzie poprzez sprzedaż pakietów akcji inwestorom. Pozyskany kapitał wykorzystują do prowadzenia i rozwijania działalności bez konieczności zaciągania dłużu. W zamian za przywilej publicznej sprzedaży akcji, spółki są zobowiązane do ujawniania informacji o swoim stanie i dawania udziałowcom prawa do decydowania o sposobie ich prowadzenia.

Ceny akcji są napędzane przez wiele czynników, ale ostatecznie cena w danym momencie jest zależna od podaży i popytu w tym momencie na rynku. Możemy wyróżnić trzy kategorie czynników oddziałyujących na ceny akcji: czynniki fundamentalne, czynniki techniczne oraz sentyment rynkowy.

Czynniki fundamentalne napędzają ceny akcji w oparciu o zyski spółki i rentowność z produkcji i sprzedaży towarów i usług.

Czynniki techniczne odnoszą się do historii cen akcji na rynku, odnosząc się do wzorców wykresów, pędu i czynników behawioralnych handlowców i inwestorów.

Sentyment rynkowy jest badany przez stosunkowo nową dziedzinę finansów behawioralnych. Wychodzi ona z założenia, że rynki przez większość czasu są nieefektywne, a nieefektywność ta może być wyjaśniona przez psychologię i inne dyscypliny nauk społecznych. Sentyment rynkowy odnosi się do psychologii uczestników rynku, indywidualnie i zbiorowo. Sentyment rynkowy jest często subiektywny i stronniczy. Na przykład, można dokonać solidnej oceny perspektyw wzrostu akcji w przyszłości, i co więcej, przyszłość może nawet potwierdzić nasze prognozy, ale w międzyczasie rynek może krótkowzrocznie rozwodzić się nad jedną wiadomością, która utrzymuje ceny akcji sztucznie wysoko lub nisko.

1.3. Definicja problemu

Osoby grające na giełdzie są zainteresowane przewidywaniem przyszłych cen akcji. Zakłada się, że algorytmy sztucznej inteligencji mogą wspomóc proces decyzyjny inwestowania. Co więcej, skuteczność tych algorytmów może zależeć od danych wejściowych. Wykorzystanie danych giełdowych przynosi mało zadowalające rezultaty. Sytuacja zaczyna wyglądać inaczej, gdy jako wejście dla algorytmu rozpatrujemy dane powiązane z giełdą, których źródłem jest platforma społecznościowa, przykładowo *Twitter*. Bezsprzeczny jest, że istnieje zależność między nastrojem autora, a charakterem i doborem słów w jego wypowiedzi. W projekcie porównujemy skuteczność przewidywania w zależności od wykorzystania, bądź nie, danych z sentymentem. Skupiamy się również na zbadaniu korelacji między sentymentem społecznym, a cenami aktywów finansowych takich jak akcje, czy kryptowaluty.

1.4. Cel pracy

Celem naszego projektu jest porównanie skuteczności przewidywania przez model sieci neuronowej przyszłych cen aktywów w zależności od dostarczonych danych. Dane pochodzą z giełdy lub z giełdy i z platformy społecznościowej *Twitter*. Celem, który wynika z poprzedniego, jest stworzenie systemu wspierającego proces decyzyjny obrotu akcjami pod postacią przeglądarkowej aplikacji użytkownika. Aplikacja ta prezentuje kluczowe kroki działania naszego systemu, takie jak: tworzenie zbioru treningowego, trening sieci, test sieci. Oczywistym jest, że chcemy, by stworzony przez nas model miał skuteczność wyższą niż 50% (skuteczność na poziomie 50% osiągają losowe algorytmy)

1.5. Przegląd rozwiązań

Rozwiązania przedstawionego problemu różnią się w kilku płaszczyznach: wyboru i przetworzenia zbioru danych, wyboru modelu, bądź też algorytmu przetwarzającego te dane. Poniżej przedstawiamy alternatywne podejścia.

1.5.1. Predicting Stock Movement Using Sentiment Analysis of Twitter Feed with Neural Networks [3]

Autorzy przewidują cenę aktywa na przyszły dzień. Na wejściu sieci neuronowej do uczenia wykorzystują następujące dane:

- indeks *DJIA* oraz historyczne dane giełdowe firmy *Apple*. Dane te są pobierane przy użyciu serwisu *Yahoo Finance*
- dane z *Twittera* dostępne z bazy *Sentiment 140* dostępnej poprzez serwis *Kaggle*.

Tak przygotowane tweety następnie wykorzystują do trenowania modeli. Najlepsze wyniki uzyskali dla maszyny wektorów nośnych (*SVM*). W celu przewidzenia ceny zamknięcia, autorzy wykorzystali dwa następujące algorytmy:

- drzewo regresji wspomaganej (*Boosted Regression Tree*)
- sieć neuronowa z perceptronem wielowarstwowym (*multilayer perceptron neural network*)

Autorzy szukają dokładnej ceny aktywa, więc nie przedstawiają skuteczności jako liczba poprawnie przewidzianych spadków czy wzrostów. Prezentują za to średnią kwadratową błędów (*RMSE*). Dla tweetów z frazą *AAPL* (symbol giełdowy firmy *Apple*) jest ona na poziomie 1.37\$ dla drzewa regresji wspomaganej i na poziomie 0.98\$ dla sieci neuronowej z perceptronem wielowarstwowym. Przy ówczesnej cenie akcji firmy *Apple* równej 118\$ jest to zadowalający wynik.

1.5.2. Stock Prediction Using Twitter Sentiment Analysis [5]

Autorzy szukają korelacji między sentymentem społecznym *public sentiment*, a sentymentem giełdowym *market sentiment*. Na wejściu sieci neuronowej do uczenia wykorzystują następujące dane:

- indeks *DJIA*. Dane te są pobierane przy użyciu serwisu *Yahoo Finance*
- tweety pochodzące z bazy danych Uniwersytetu Stanford *SNAP*. Zostały one przetworzone w opisany poniżej sposób

Autorzy opracowali własną listę słów na podstawie znanego kwestionariusza profilu stanów nastroju - *POMS*. *POMS* jest kwestionariuszem psychometrycznym, który prosi osobę o ocenę jej aktualnego nastroju poprzez udzielenie odpowiedzi na 65 różnych pytań w skali od 1 do 5. Przykładowo odpowiadający ma ocenić w skali od 1 do 5, jak bardzo czuje się dziś **spięty**, jak bardzo czuję się dziś **bezradny** etc.. Te 65 słów jest następnie mapowane na 6 standardowych nastrojów *POMS*: napięcie, depresję, złość, vigor, zmęczenie i zmieszanie. Aby zautomatyzować tę analizę tweetów, lista słów musi być znacznie powiększona. Autorzy rozszerzyli listę poprzez uwzględnienie wszystkich powszechnie występujących synonimów dla bazowych 65 słów. Użyli do tego zasobu leksykalnego do badania opinii *SentiWordNet* oraz standardowego tezaurusa.

Następnie autorzy wyliczyli wynik dla każdego słowa na podstawie liczby jego dopasowań w tweetach z danego dnia. Potem dwukrotnie mapowali otrzymane wyniki, by ostatecznie otrzymać wartości dla czterech wybranych stanów nastroju.

Autorzy, w celu zbadania rzeczywistej korelacji między sentymentem, a cenami aktywów, wypróbowali 4 różne algorytmy uczenia: regresję liniową (*Linear Regression*), regresję logistyczną (*Logistic Regression*), maszynę wektorów nośnych (*SVM*) i samoorganizujące się rozmyte sieci neuronowe.

Ich wyniki wskazują również, że *SOFNN* radzi sobie najlepiej spośród wszystkich algorytmów, osiągając 75,56% dokładności. Czy wykorzystanie sieci neuronowej *LSTM* może dać lepsze wyniki niż wykorzystanie samoorganizującej się rozmytej sieci neuronowej (*SOFNN*)?

1.5.3. A hybrid model integrating deep learning with investor sentiment analysis for stock price prediction [2]

Autorzy proponują model hybrydowy, który łączy głębokie uczenie z modelem analizy sentymentu do przewidywania cen akcji. Stosują oni model konwolucyjnej sieci neuronowej (*CNN*)

do klasyfikacji ukrytego sentymentu inwestorów, pozyskanego z wpisów na forum giełdowego. Następnie wykorzystują model sieci neuronowej *LSTM* do analizy wskaźników technicznych z rynku akcji oraz analizy sentymentu. Co ciekawe, autorzy używają danych giełdowych z sześciu różnych branż, by pokazać, że wybrana sieć *LSTM* działa dla różnych kontekstów, nie tylko finansowego. W przewidywaniu sentymentu przy użyciu sieci *CNN* uzyskali 82% skuteczność podczas treningu.

1.5.4. Using Neural Networks to Forecast Stock Market Prices [4]

Autor dokonuje gruntownego przeglądu metod analitycznych służących do przewidywania przyszłych cen aktywów. Wspomina o ciekawej teorii chaosu, która analizuje proces przy założeniu, że jego pewna część jest deterministyczna, a pozostała losowa. Opisuje proces wykorzystania sieci neuronowej w kontekście przewidywania, od przygotowania danych, przez trening i organizację, do wyników.

1.5.5. Stock Market Prediction Using Artificial Neural Networks [7]

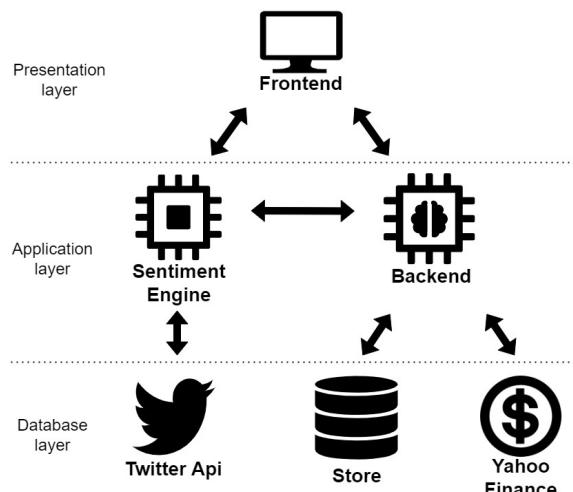
Autorzy wykorzystują sieci neuronowych z propagacją wsteczną do przewidywania dziennej ceny indeksu złożonego *Shanghai Stock Exchange*. W rezultacie swojej pracy stwierdzają, że wspomniany indeks jest przewidywalny w krótkim okresie. Przy użyciu sieci przewidują jego najwyższą, najniższą i zamykającą wartość.

1.5.6. Using Deep Learning to Develop a Stock Price Prediction Model Based on Individual Investor Emotions [1]

Autorzy przedstawiają koncepcję systemu przewidywania akcji opartego na emocjach inwestorów indywidualnych. Ceny aktywów finansowych przewidują wykorzystując głęboką sieć neuronową (*DNN*).

2. ARCHITEKTURA SYSTEMU

W celu rozwiązywania problemu przewidywania cen aktywów finansowych stworzyliśmy schemat architektury, który z czasem skrystalizował się w poniższym rysunku. Rezultatem pracy nad projektem jest przeglądarkowa aplikacja użytkownika, która komunikuje się z serwisami po stronie *backendu* i pozwala na stworzenie zbioru treningowego, trening oraz test sieci.



Rys. 2.1. Schemat architektury systemu

2.1. Realizacja architektury

Architektura mikroserwisów, to styl tworzenia architektury aplikacji komputerowych, który implementuje wzorzec architektury zorientowanej na usługi. Architektura ta aranżuje aplikację jako zbiór luźno połączonych ze sobą niewielkich serwisów komunikujących się poprzez lekkie protokoły komunikacyjne (*HTTP, HTTPS, DNS, FTP*).

Aby zrealizować powyższy model, zdecydowaliśmy się na architekturę mikroserwisów z wyraźnym podziałem na trzy warstwy: prezentacji (aplikacja użytkownika), logiki biznesowej (aplikacji) oraz danych. Aplikacja użytkownika pozwala sterować siecią neuronową oraz serwisem przetwarzającym dane z sentymentem z poziomu przeglądarki. Za logikę biznesową odpowiada serwis o nazwie *Backend*. Zawiera on w sobie sieć neuronową, obsługuje komunikację z warstwą danych (ma dostęp do API serwisu *Yahoo Finance*) oraz przetwarza żądania aplikacji użytkownika. Serwis *Sentiment engine* ma dostęp do API *Twittera* i odpowiada za pobieranie oraz przetwarzanie tweetów. *Store* to po prostu folder zawierający pliki z przygotowanymi wcześniej danymi z sentymentem.

Platforma *Twitter* udostępnia rozbudowane *API*, które można łatwo przetestować dzięki narzędziu *Twitter API v2 calls*. Pozwala ono pobierać informacje o:

- tweetach historycznych
- wybranym użytkowniku i jego tweetach

- liczbie danych *tweetów*

Co ważne, możliwe jest sensowne filtrowanie pobieranych informacji (np. po zakresie dat) oraz wyciąganie dodatkowych parametrów, przykładowo przypisów kontekstowych (ang. *context annotations*), czy publicznych metryk (ang. *public metrics*).

2.2. Wykorzystanie protokołu HTTP

W ramach działania całego serwisu niezbędna jest komunikacja między poszczególnymi podprojektami. Podprojekty te wymieniają informacje takie jak pobrane dane z giełdy, czy dostępne wytrenowane modele. Ponadto użytkownik końcowy korzystając z aplikacji uruchamia różne procesy jak przykładowo rozpoczęcie uczenia, czy przewidywania przez sieć neuronową. W celu realizacji powyższej komunikacji wykorzystaliśmy najpopularniejszy protokół w sieci *WWW*, czyli *HTTP*.

Jako że logikę projektu zaimplementowaliśmy w języku programowania *Python*, zdecydowaliśmy się użyć *frameworka webowego Flask*. *Flask* służy do budowania aplikacji webowych. Jest lekki i nie zawiera zbędnych narzędzi i mechanizmów, takich jak ORM czy obsługa formularzy. Minimalizm architektury wraz z możliwością integrowania się z dodatkowymi bibliotekami pozwala na łatwe dostosowanie środowiska. Jest zalecany do niewielkich projektów (źródło). Można w nim łatwo obsługiwać napływające żądania. Wystarczy oznaczyć daną metodę dekoratorem:

```
@app.post("/train")
```

2.3. Infrastruktura

Rozwijanie aplikacji składającej się z kilku modułów wymaga środowiska, które pozwala ją sprawnie uruchomić. W tym celu zdecydowaliśmy się wykorzystać platformę konteneryzacji *Docker*.

Docker to narzędzie, które uruchamia kontenery. Kontener to instancja obrazu *Docker*. A obraz *Docker* zawiera wykonywalny kod źródłowy aplikacji, a także wszystkie narzędzia, biblioteki i zależności, których kod aplikacji wymaga do uruchomienia jako kontener. Dzięki kontenerom możemy uruchomić dodatkowy, odizolowany system operacyjny z gotową do działania aplikacją. Kontener nie emuluje całej warstwy sprzętowej, dzięki niemu otrzymujemy pełnoprawny system operacyjny wraz ze zdefiniowanym procesem aplikacji.

Podsumowując, *Docker* jest otwarto-źródłowym oprogramowaniem pozwalającym umieścić aplikację oraz jej zależności (biblioteki, pliki konfiguracyjne, lokalne bazy danych itd.) w lekkim, przenośnym, wirtualnym kontenerze.

W celu stworzenia obrazów *Docker* w każdym projekcie dodaliśmy plik *Dockerfile*. Przykładowy plik należący do projektu z logiką i siecią wygląda jak w listingu 7.2.

Warto zwrócić uwagę na rozbudowaną komendę *RUN*. Dzięki niej, w momencie tworzenia środowiska dla projektu, nie musimy pobierać już wcześniej pobranych modułów, więc tworzenie kontenera zajmuje mniej czasu i łącza internetowego.

Ponadto, realizując projekt zależało nam, by przyszły użytkownik oprogramowania lub osoba zainteresowana rozwojem naszego pomysłu mogła w łatwy sposób uruchomić wszystkie moduły projektu. Utworzyliśmy plik *docker-compose.yaml* pozwalający jedną komendą uruchomić wszystkie obrazy. Plik wygląda jak w listingu 7.3.

Jedną z większych zalet automatyzowania uruchamiania poszczególnych podprojektów jest to, że później projekt można z łatwością uruchomić przy użyciu platformy oprogramowania *Kubernetes*. Służy ona do zarządzania zadaniami i serwisami uruchamianymi w kontenerach oraz umożliwia ich deklaratywną konfigurację i automatyzację. Dużą zaletą jest również bardzo łatwe współdzielenie danych między kontenerami. Służą do tego tzn. wolumeny (ang. *volumes*). W powyższym pliku *docker-compose.yaml* definicja wiązania folderu z systemu pliku użytkownika komputera do systemu plików kontenera odbywa się w następującej linijce:

Listing 2.1: fragment pliku *docker-compose.yaml* pozwalający na współdzielenie danych

```
volumes:  
  - sentimental-data:/src/data:rw
```

gdzie najpierw określamy nazwę folderu w docelowym kontenerze, a następnie ścieżkę do zawartości, którą chcemy udostępnić.

3. PRZYGOTOWANIE DANYCH

Pierwszym krokiem w implementacji naszego projektu było przygotowanie danych do analizy. Dobre wybranie struktury używanych danych znacząco ułatwia ich przetwarzanie. W poniższych sekcjach przedstawiamy nasze podejście.

3.1. Przygotowanie danych z giełdy

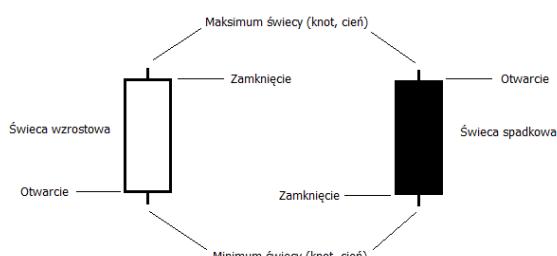
W celu wytrenowania sieci neuronowej potrzebne było utworzenie odpowiednio dużej bazy danych, która zawiera giełdowe ceny aktyw finansowych. Dane te dostępne są do pobrania za darmo w serwisach internetowych takich jak *Yahoo finance*, czy *AlphaVantage*.

3.1.1. Format danych

Najpopularniejszym sposobem przechowywania danych giełdowych jest tabela. Wiersz tabeli oznacza pojedynczy okres. Kolumny natomiast to informacje odnośnie danego okresu. Pierwszy wiersz zawiera tytuły kolumn. Najczęściej spotykane informacje odnośnie akcji to:

- Cena otwarcia (*Open*)
- Najwyższa cena w okresie (*High*)
- Najniższa cena w okresie (*Low*)
- Cena Zamknięcia (*Close*)

Na podstawie powyższych informacji tworzone są wykresy *Świec Japońskich* przedstawionych na rysunku 3.1. Wykresy te w lepszym stopniu obrazują zachowanie ceny aktywa w danym przedziale czasu, w przeciwieństwie do zwykłej średniej ceny aktywa. Niektóre serwisy oferują dodatkowe informacje takie jak **liczba transakcji**(*Volume*) oraz **skorygowana cena zamknięcia**(*Adj. Close*).



Rys. 3.1. Świece japońskie
źródło: <https://trading-academy.pl/co-to-sa-swiece-japonskie/>

Tabela 3.1. Przykład formatu danych akcji firmy Amazon. interwał = 1 dzień.
źródło <https://finance.yahoo.com/quote/AMZN/history?p=AMZN>

Date	Open	High	Low	Close	Adj Close	Volume
2021-11-15	176.850006	179.694000	176.290497	177.283997	177.283997	58594000
2021-11-16	176.949997	178.824997	176.257507	177.035004	177.035004	44342000
2021-11-17	178.235992	179.362503	177.267502	177.449997	177.449997	512060

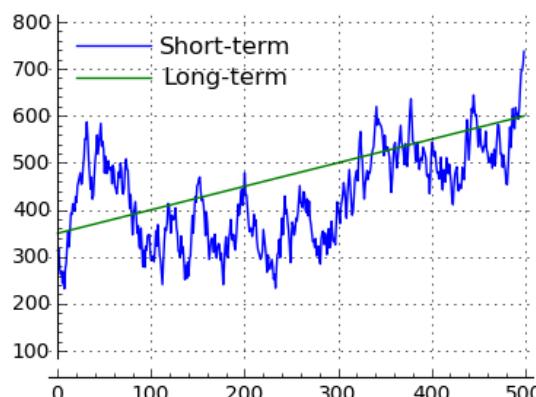
W tabeli 3.1 przedstawiliśmy przykładowy zbiór danych na temat akcji dostępny do pobrania z serwisu *Yahoo*. Warto zauważyc, że w podanej tabeli nie mamy informacji na temat waluty. Trzeba ją wywnioskować z rynku, na którym sprzedawana jest dana akcja. W naszym projekcie zdecydowaliśmy się wykorzystać format danych zgodny z tabelą 3.1. Interesuje nas najbardziej giełda Amerykańska, więc walutą którą będziemy się posługiwać jest dolar.

3.1.2. Interwał

Dane na temat akcji można zapisywać z różnym interwałem. Ze względu na długość interwału możemy podzielić na 2 rodzaje:

- dzienny - 1 min., 5 min., 15 min., 30 min., 60 min.
- długoterminowy - 1 dzień, 1 tydzień, 1 miesiąc

Interwały dzienne sprawdzają się najlepiej przy handlu krótkoterminowym, w którym o zysku lub stracie decydują pojedyncze minuty. Problemem interwałów dziennych jest ilość danych, którą serwisy muszą przechowywać. Danych z okresem 1 minuty zajmują 1440 razy więcej miejsca niż te z interwałem 1 dnia. Z tego powodu serwisy takie jak *Yahoo* nie oferują dostępu do danych z interwałem mniejszym niż 1 dzień. Natomiast inne jak *AlphaVantage* udzielają dostępu tylko do 2 lat wstecz. Interwały długoterminowe wykorzystywane są najczęściej przy analizie zmian cen aktywów na przestrzeni miesięcy i lat. Najczęściej spotykanym interwałem jest 1 dzień, ponieważ gwarantuje on dobry kompromis między szczegółowością danych, a ich ilością. Pozwala śledzić najnowsze trendy jak i patrzeć na dane z perspektywy czasu. Okresy tygodniowe i miesięczne mogą być przydatne przy analizowaniu cen aktywów na przestrzeni dekad. Dzięki tak rzadkiemu próbkowaniu z łatwością można dostrzec ogólny trend i nie skupiać się na lokalnych spadkach, czy wzrostach. Różnice w tych podejściach możemy zobaczyć na rysunku 3.2



Rys. 3.2. Inwestowanie dugo i krótko terminowe
źródło: https://arachnoid.com/equities_myths/index.html

3.1.3. Sposób pozyskiwania danych

Jak wykazaliśmy w poprzednich sekcjach, istnieje wiele serwisów, które udostępniają dane giełdowe. Zazwyczaj różnią się one dwoma elementami.

Po pierwsze, sposobem pozyskiwania danych. Można pobierać je ręcznie, albo poprzez programistyczny interface (*API*). Po drugie, serwisy różnią się dostępnością specjalnych aktywów takich jak *kryptowaluty*. Wybierając sposób pobierania danych, rozważaliśmy następujące podejścia:

- ręczne pobieranie danych
- automatyzacja ręcznego pobierania danych (*Scraping*)
- skorzystanie z API

Ręczne pobieranie danych giełdowych jest sposobem, który wymaga najmniej przygotowania i jest najbardziej naturalny dla użytkowników przyzwyczajonych do korzystania z internetu. Jego problemem jest linowa skalowalność czasu wykonania tego rozwiązania, pobranie 100 arku-

szy danych o akcjach zajmie 100 więcej niż pobranie jednego. Drugim problemem jest konieczność manualnego pobierania najbardziej aktualnych danych. Dlatego ta metoda sprawdza się doskonale przy pobieraniu niewielu zestawów danych. W naszym projekcie użyliśmy tej metody na początku w celu testowania sieci neuronowej.

Naszym drugim pomysłem była automatyzacja ręcznego pobierania danych przy użyciu narzędzia pozwalającego sterować przeglądarką. W tym celu chcieliśmy skorzystać z *Fremwarka Selenium*. Ten sposób rozwiązuje problem skalowalności ręcznego pobierania danych. Ma natomiast wysoki próg wejścia, ponieważ wymaga czasochłonnego zbadania struktury strony i przygotowania programu nawigującemu po przeglądarce w celu pobrania arkuszy danych. Jednak największym problemem takiego rozwiązania była potrzeba aktualizowania programu, gdy zdarzyły się niezależne od nas zmiany na stronie serwisu. Choć pobieranie publicznych danych z internetu jest zazwyczaj legalne, serwisy często wprowadzają własne ograniczenia zabraniające *scrapingu*.

Ostatnim rozwiązaniem było skorzystanie z interfejsu programistycznego. Takie *API* pozwala w łatwy sposób z poziomu programu odwołać się do zasobów przechowywanych przez serwis. Jest to najbardziej naturalny dla programistów sposób komunikacji z serwisem. Problemem takiego rozwiązania jest konieczność zapłaty za dostęp do bardziej szczegółowych danych jak akcje z interwałami dziennymi. Większość serwisów wprowadza także limit na ilość danych pobieranych w danym czasie. Rozważając powyższe opcje zdecydowaliśmy się skorzystać z *API*.

Alpha Vantage

W początkowej fazie projektu naszą potrzebą był darmowy dostęp do danych zarówno z interwałami dziennymi jak i długoterminowymi. Chcieliśmy mieć także możliwość pobierania informacji zarówno o cenach akcji jak i *kryptowalut*. Zależało nam także na jak najdłuższym zakresie danych. Takie założenia przyczyniły się do wybrania przez nas serwisu *Alpha Vantage*. Serwis ten oferuje w darmowej wersji dostęp do ponad dwudziestu lat historycznych cen akcji w interwałach długoterminowych oraz dostęp do dwóch ostatnich lat dla interwałów dziennych. Udostępnia również dodatkowe informacje ekonomiczne jak wskaźnik inflacji w Stanach Zjednoczonych. Problemem tego serwisu jest niespójny interfejs programistyczny do pobrania danych. Komunikacja z serwisem odbywa się za pomocą zapytań HTTP czyli *Hypertext Transfer Protocol*. Adresy sieciowe różnią się w zależności od interwału jak w przykładzie 3.1. Co utrudnia parametryzację zapytań.

Przykład 3.1.

url = <https://www.alphavantage.co>

url?function=TIME_SERIES_INTRADAY&symbol=IBM&interval=5min

url?function=TIME_SERIES_DAILY_ADJUSTED&symbol=IBM

Serwis

Problemy przedstawione w powyższej sekcji skłoniły nas do stworzenia serwisu, który miałby ujednolicić przedstawione wyżej *API*. Nasz serwis miał także adresować problem stronicowania, który pojawił się dla danych z interwałami dziennymi. Dla takich interwałów *Alpha Vantage* pozwalał na pobieranie danych maksymalnie z zakresu miesiąca na każde zapytanie. Serwis ten

nazwaliśmy *Stock-scrapper*. Stworzyliśmy go w języku *Python*. Do obsługi zapytań *HTTP* użyliśmy *Frameworka Django*. *Stock-scrapper* udostępniał dwa spójne *Endpointy*:

- `baseAddress/stock/`
`?symbol=symbol&interval=interwał&start_date=data_startu&end_date=data_końca`
- `baseAddress/crypto/`
`?symbol=symbol&interval=interwał&start_date=data_startu&end_date=data_końca`

Gdzie:

- symbol aktywa (*ticker*)
- interwał - jest jednym z dostępnych:
 - 1 min.
 - 5 min.
 - 15 min.
 - 60 min.
 - *daily*
 - *weekly*
 - *monthly*
- *data_startu* - pierwszy dzień danych mierzony od 00:01
- *data_końca* - ostatni dzień danych mierzony do godziny 00:00

Endpointy zwracały dane w formacie zgodnym z podanym w tabeli 3.1 w pliku z rozszerzeniem *csv*.

yfinance

W trakcie rozwoju projektu odkryliśmy, że rzadko korzystamy z interwałów innych niż jednodniowe. Podczas rozwijania aplikacji zauważylismy także problem pobierania danych giełdowych i ich przechowywania przez dłuższy czas. Nierzadko firmy decydują się na podzielenie swoich akcji na mniejsze części. Takie działanie określa się angielskim słowem *split*, czyli podział. *Split* 20:1 oznacza, że każda akcja podzielona jest na dwadzieścia nowych, co jest równoznaczne z dwudziestokrotnym zmniejszeniem ceny pojedynczej akcji.

Przykład 3.2. 9 marca 2022 *Amazon.com* dokonał podziału 20:1 zmniejszając wartość pojedynczej akcji z 2447 USD do około 122 USD.

Jak że firma nie traci na wartości po podziale, nie uwzględniając oczywiście reakcji inwestorów, potrzebna jest korekcja historycznych cen, tak by odpowiadały obecnej liczbie akcji. Podczas *Splitu* z przykładu 3.2, wszystkie historyczne ceny akcji firmy *Amazon.com* zostały podzielone dwudziestokrotnie, natomiast wolumen (*Volume*) został pomnożony dwudziestokrotnie. Spowodowało to dezaktualizacje naszego obecnego zbioru danych. Ta zmiana w podejściu oraz zauważony problem pchnął nas do pobierania danych giełdowych zawsze podczas trenowania sieci. Jako że zdecydowaliśmy się skupić tylko na interwale jednodniowym, wybraliśmy do tego celu bibliotekę *python yfinance*. Biblioteka ta oferuje spójne *API* umożliwiające pobieranie danych z serwisu *Yahoo Finance* z poziomu języka *python*. Biblioteka jest darmowa. Jej implementacja korzysta z *API Yahoo Finance* co daje dostęp do szerokiej gamy danych na temat aktywa, takich jak sytuacja finansowa firmy, czy ostatnie wiadomości na jej temat. Jedna funkcja pozwala pobrać zarówno dane na temat akcji jak i kryptowalut. Naszym początkowym zmartwieniem był wpływ czasu pobierania danych na ogólny proces treningu. Lecz po kilku testach, zaobserwowaliśmy,

że pobranie danych z ostatnich 10 lat dla jednego aktywa trwa zaledwie 0.5 sekundy. Powodem rezygnacji z tej biblioteki na początku był brak dostępu do interwałów dziennych, które uważaliśmy za wartościowe w początkowych fazach projektu.

3.2. Przygotowanie danych z Twittera

Jako źródło danych do analizy sentymetru rozpatrywaliśmy dwie platformy społecznościowe: *Facebook* oraz *Twitter*. Zdecydowaliśmy się wybrać *Twittera*, ponieważ posiada rozbudowane API, które pozwala na wyrafinowane filtrowanie i pozyskanie szczegółowych informacji o danym tweecie. Ponadto, istnieje sporo otwartych zbiorów danych utworzonych na podstawie tweetów, tematycznie powiązanych z giełdą.

3.2.1. Pozyskanie tweetów

Tweety, czyli wpisy na platformie społecznościowej *Twitter* mogliśmy pozyskać na kilka sposobów:

- bezpośrednio z *Twittera* przy po pomocy udostępnionego API
- korzystając z gotowych zbiorów danych

Naszym celem było pozyskanie jak największej liczby tweetów, by dostarczyć jak najwięcej danych na wejście sieci neuronowej. Dlatego uzyskaliśmy dostęp akademicki do *Twitter API v2..* Dostęp taki pozwala na pobieranie 10 milionów tweetów miesięcznie, z limitem do 300 żądań na 15 minut. Ponadto zależało nam na tym, by dane były jak najświeższe.

Stworzyliśmy funkcję pobierającą aktywa finansowe obecne w pliku w formacie *.json* z listą wpisów, w których kluczem jest symbol (*token*), a wartością nazwa firmy. Funkcją ta pobiera tweety, które zawierają symbol lub nazwę firmy i zapisuje w folderze. Zapisanych plików dla przeciętnego aktywa jest od 2 do 3 tysięcy (w pliku znajduje się maksymalnie 500 tweetów - tyle tweetów znajduje się w pliku json zwróconym jako odpowiedź na pojedyncze żądanie). Następnie pliki są czytane i dla każdego tweetu wyliczany jest jego sentyment.

3.2.2. Dostępne zbiorы danych

Przeglądając rozwiązania natrafiliśmy na kilka interesujących, gotowych zbiorów danych. Zbiory zawierające dane najbliższe naszym potrzebom to:

- *STOCK MARKET TWEETS DATA*: to zbiór 943672 tweetów z okresu od 9.04.2020 do 16.07.2020. Powstał poprzez pobieranie tweetów z tagami *S&P 500 (#SPX500)* oraz tagami 25 największych spółek z indeksu *S&P 500* oraz tagu *Bloomberg (#stocks)*. 1300 z 943672 tweetów zostało ręcznie sklasyfikowanych tzn. przypisanych do klasy pozytywny, neutralny lub negatywny.
- *Financial Tweets*: to zbiór zawierający 28 tysięcy tweetów stworzonych przez 30 najpopularniejszych influencerów, takich jak *MarketWatch*, *business*, *YahooFinance*. Tweety pochodzą z okresu od 18.07.2018 do 23.02.2018.

Zbiory te zawierają relatywnie mało tweetów (stworzony przez nas finalny zbiór ma ponad 20 milionów).

3.2.3. Filtrowanie tweetów

Twitter pod koniec roku 2019 miał 330 milionów aktywnych użytkowników. W każdej sekundzie wysyłanych jest średnio 6 tysięcy tweetów, co odpowiada ponad 350 tysiącom tweetów na minutę, 500 milionom tweetów dziennie i około 200 miliardom tweetów rocznie. Co ciekawe, poza

dzieleniem się zabawnymi treściami takim jak *memy*, Twitter jest w rzeczywistości przydatny na wiele innych sposobów. Jest najczęściej używany do uzyskania konkretnych odpowiedzi i porad bezpośrednio od kompetentnych ludzi oraz dyskusji.

Interesujący jest fakt, że 10% użytkowników odpowiada za publikowanie 92% tweetów. Jest to podręcznikowy przykład zasady Pareto. Czy w związku z tym, powinniśmy zmienić objętą strategię - pobieraj wszystkie tweety zawierające dane słowo, na strategię, w której skupiamy się na najczęściej publikujących bądź najbardziej wpływowych użytkownikach? Postanowiliśmy pozostać przy pierwotnie obranej strategii, ponieważ dzięki niej możemy zgromadzić więcej tweetów. Wracając do statystyk, okazuje się, że wg statystyk Twittera mniej niż 5% kont twitterowych to boty. Elon Musk, uważa, że aż do 20% kont może należeć do botów. Firma Similarweb raportuje, że od 20% do 29% zawartości generowana jest przez boty. Okazało się również, że tylko 19% prawdziwych, uwierzytelnionych użytkowników Twittera w USA generuje codziennie treści.

W świetle powyższych danych zadaliśmy sobie pytanie, czy lepiej przetworzyć tweety pochodzące z wszystkich kont, czy też tylko z kont rzeczywistych użytkowników. Jeśli bralibyśmy tweety z wszystkich kont, moglibyśmy podać na wejście sieci neuronowej więcej danych. Jeśli przefiltrowalibyśmy tweety, tak by mieć pewność, że są pisane przez rzeczywistych użytkowników, a nie generowane automatycznie, to uzyskalibyśmy większą wiarygodność podczas wyliczania sentymenu. Większa wiarygodność wynikałaby z faktu, że popularność tweetów napisanych przez boty zwiększa inne boty (wynika to z naszych obserwacji pobranych tweetów).

Wybraliśmy większą wiarygodność i mniejszą liczbę danych. W kolejnym paragrafie opisujemy w jaki sposób przesiewamy tweety.

W celu odfiltrowania tweetów stworzyliśmy prosty dwukrokovy algorytm. Najpierw sprawdź, czy wśród metryk tweetu są wartości większe od zera. Jeśli tak dodaj je na listę z metrykami. Jeśli lista zawiera przynajmniej dwie metryki to znaczy, że z tweetem wystąpiły przynajmniej dwie różne interakcje, więc prawdopodobieństwo, że tweet został stworzony przez bota jest dużo mniejsze (chcemy przetworzyć taki tweet). Poniżej znajduje się pseudokod opisanego algorytmu:

Algorithm 1 Sprawdzanie czy tweet ma znaczenie

```
1: function SPRAWDZCZTTWEETMAZNACZENIE
2:   Initialize czyTweetMaZnaczenie = false
3:   Initialize M = metrykiTweetu
4:   Initialize L = newList           ▷ lista, która będzie zawierać znaczące metryki
5:   for each m ∈ M do          ▷ krok 1
6:     if m > 0 then
7:       L.dodaj(m)
8:     end if
9:   end for
10:  if L.dlugosc > 1 then          ▷ krok 2
11:    czyTweetMaZnaczenie = true
12:  end if
13:  return czyTweetMaZnaczenie
14: end function
```

3.3. Analiza sentymenu

Analiza sentymenu to wykorzystanie przetwarzania języka naturalnego, analizy tekstu, lingwistyki komputerowej i biometrii do systematycznego identyfikowania, wyodrębniania, określania ilościowego i badania stanów afektywnych i informacji subiektywnych.

W definicji bliższej biznesowi możemy ją zdefiniować jako kontekstową eksplorację tekstu, która identyfikuje i wydobywa subiektywne informacje w materiale źródłowym, pomagając przedsiębiorstwu w zrozumieniu społecznego sentymentu jakim cieszy się jego marka, produkt lub usługa. W takiej sytuacji najczęściej analizuje się wpisy, komentarze na temat danej marki, czy po prostu konwersacje online z klientem.

Wychodzimy z założenia, że słowa, których używamy mają znaczenie, ponieważ odzwierciedlają stan emocjonalny autora lub jego intencje. Nie zagłębiając się w specyfikę użytkowania platformy *Twitter*, przyjęliśmy założenie, że autorzy *tweetów* wyrażają w nich swoje prawdziwe emocje i nastawienie, a liczba *tweetów*, w których autor manipuluje i sztucznie przedstawia swoje poglądy, czy emocje, jest marginalna.

Wybierając bibliotekę do analizy sentymentu przejrzaliśmy działanie i specyfikę dwóch najpopularniejszych bibliotek: *VADER* oraz *Flair*.

3.3.1. Działanie biblioteki *VADER*

Biblioteka *VADER* opiera się na analizatorze sentymentu, który ma zdefiniowane reguły. Przypisuje on pozytywną lub negatywną ocenę pewnym słowom (np. słowo *straszny* ma negatywne powiązanie). Zwraca uwagę na negację, jeśli istnieje, i zwraca wartości oparte na tych słowach. Do jej największych zalet należy to, że jest prosta i bardzo szybka. Niestety, gdy analizowane zdanie jest długie, pojawia się więcej neutralnych słów, a zatem ogólny wynik ma tendencję do zbiegania w kierunku neutralnym. Poza tym często źle interpretuje sarkazm i żargon. Pierwsza wada nie jest przeszkodą, bo *weetety* statystycznie mają długość 33 znaków. Druga wada może istotnie wpływać na jakość analizy, ponieważ zakładamy, że *weet* jest pisany tak, by był zrozumiany przez innych odbiorców, więc jeśli analizator nie może go zrozumieć, źle go ocenia.

Ze względu na szybkość analizowania, zdecydowaliśmy się wybrać tę bibliotekę.

3.3.2. Działanie biblioteki *Flair*

Flair jest wstępnie wytrenowanym modelem opartym na osadzaniu. Oznacza to, że każde słowo jest reprezentowane wewnątrz przestrzeni wektorowej. Słowa z reprezentacją wektorową najbardziej podobną do innego słowa są często używane w tym samym kontekście. Pozwala to na określenie sentymentu danego wektora, a tym samym danego zdania.

Flair jest znacznie wolniejszy od swoich odpowiedników opartych na regułach. Przykładowo analizowanie 1200 zdań, zajmuje bibliotece 49 sekund, podczas gdy bibliotece *VADER* 0,78 sekundy, czyli ponad 50 razy krócej.

Jednak według artykułu przedstawiającego porównanie biblioteki *Flair* i *VADER*, *Flair* radzi sobie dużo lepiej w przypadku analizowania negatywnych treści, ponieważ potrafi poprawnie ocenić podkreślenie, żargon, czy sarkazm.

3.4. Przetwarzanie pobranych tweetów

Gdy stworzyliśmy bazę *weetów* dla kilkunastu aktywów finansowych, zaczęliśmy je przetwarzać. Dla każdego pliku z pobranymi *weetami* z folderu danego aktywa wykonujemy następujące operacje:

1. otwórz plik i sczytaj z niego *weetety*
2. dla każdego *weetu* policz jego sentyment negatywny, neutralny i pozytywny
3. uśrednij sentyment dla *weetów* z danego dnia z osobna dla każdego typu sentymentu

4. zsumuj metryki *tweetów* z danego dnia: w ramach każdej metryki zsumuj liczbę polubień, komentarzy, *retweetów* i cytowań
5. do wiersza w wynikowym pliku .csv wpisz kolejno datę, sentyment oraz zsumowane wartości metryk.
6. jako rezultat otrzymasz plik .csv z ośmioma kolumnami i liczbą wierszy równą liczbie dni w zadany zakresie czasie

Poniższa tabela prezentuje wynikowy plik z sentymentem i metrykami *tweetów*.

Tabela 3.2. Przykład formatu pliku z przetworzonymi danymi z *Twittera*
źródło: opracowanie własne

Date	negative	neutral	positive	retweet_count	reply_count	like_count	quote_count
2014-01-01	0.09	0.68	0.21	179	48	207	0
2014-01-02	0.01	0.79	0.19	90	17	68	0
2014-01-03	0.03	0.88	0.08	110	15	115	0

Legenda:

negative, neutral, positive - uśrednione wartości danego sentymentu *tweetów* z dnia
retweet_count, reply_count, like_count, quote_count - metryki *tweetów* zsumowane z dnia

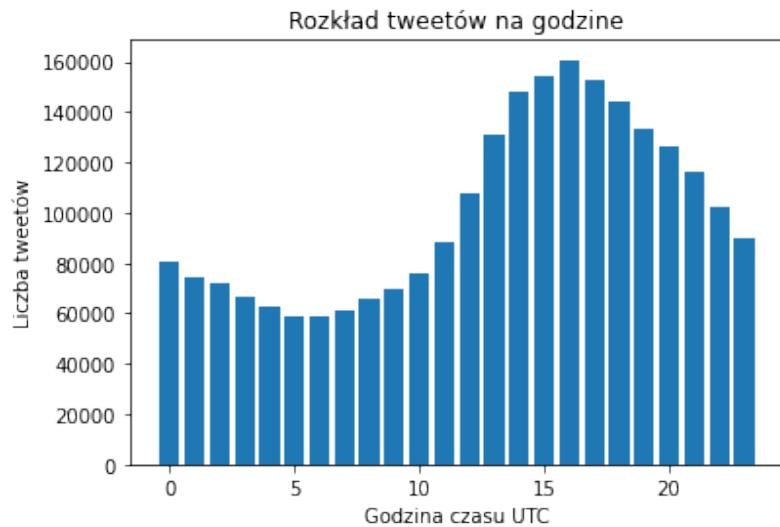
Standardowa ilość wierszy w pliku jest równa liczbie dni z zakresu, z którego pobieramy tweety. Utworzona przez nas baza danych z *tweetami* w przeważającej części dotyczy okresu od 2014.01.01 do 2022-10-01. Czyli przeciętny plik ma około 3200 linii.

3.5. Korekcja danych

Tworząc projekt natrafiliśmy na potrzebę pobierania danych z dnia bieżącego. Potrzebowaliśmy tych danych przy przewidywaniu ceny akcji na następny dzień. W przypadku pozyskiwania danych z sentymentem, problemem pobierania danych z dnia bieżącego jest brak ciągłości danych w ciągu dnia w stosunku do poprzednich dni, jak w przykładzie 3.3.

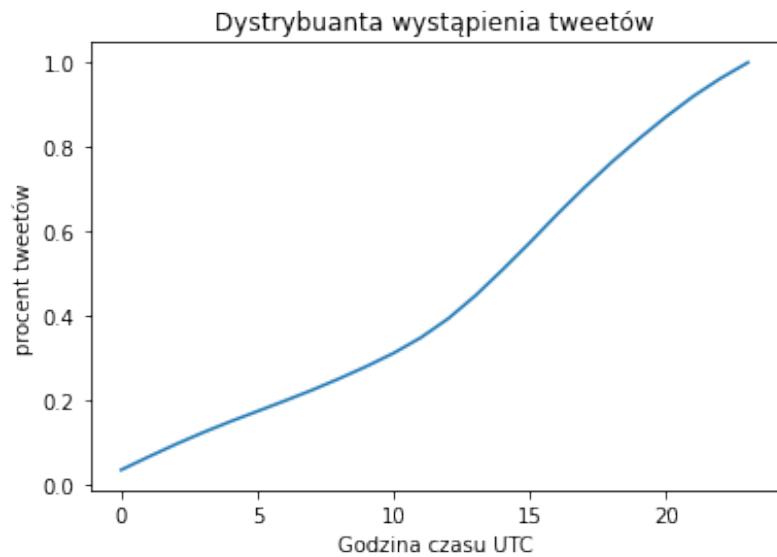
Przykład 3.3. Gdy pobieramy tweety o godzinie 12:00 jest ich 1222, natomiast o 16:00, jest ich już 3213. Dzień wcześniej, w ciągu 24 godzin pobraliśmy 4538 tweetów.

Jak podaliśmy w sekcji Przetwarzanie pobranych tweetów, wektor sentymentu zawiera oprócz uśrednionego sentymentu, zsumowane metryki *tweetów* z danego dnia. Powoduje to znaczący spadek dokładności przewidywania w przypadku niepełnych danych na temat *tweetów* danego dnia. Z tego powodu postanowiliśmy skalować sumowane metryki z dnia bieżącego na podstawie godziny ich pobrania. Najprostszym podejściem było założenie, że liczba *tweetów* rozkłada się równomiernie w ciągu całego dzień. Czyli pobierając metryki *tweetów* o 12:00 mnożylibyśmy cechy będące wynikiem sumowania z tabeli 3.2 dwukrotnie. Jednak w celu osiągnięcia największej dokładności skalowania postanowiliśmy policzyć rozkład ilości *tweetów* na podstawie godziny ich utworzenia. Wyniki naszej pracy przedstawiliśmy na wykresie 3.3.



Rys. 3.3. Rozkład tweetów, na podstawie 2402237 tweetów

Następnie na podstawie wykresu 3.3 stworzyliśmy dystrybuantę liczby *tweetów* przedstawioną na wykresie 3.4. Na jej podstawie jesteśmy w stanie w łatwy sposób skalować dane pobrane o dowolnej porze. Zakładamy przy tym, że sentyment nie jest mocno skorelowany z godziną utworzenia *tweetu*.



Rys. 3.4. dystrybuanta tweetów

4. MODEL

Sercem naszej aplikacji jest model sieci neuronowej odpowiedzialny za przewidywanie cen aktyw. Jedną z decyzji, które musielibyśmy podjąć był wybór rodzaju sieci. Obecnie znanych jest dziesiątki rodzajów sieci, które różnią się skutecznością w zależności od zastosowania. Żeby podjąć odpowiednią decyzję odnośnie wyboru zgłębiliśmy budowę i działanie poszczególnych sieci.

4.1. Budowa sieci neuronowej

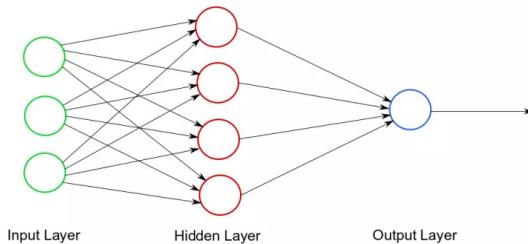
Budowa sieci neuronowych wzorowana jest na budowie neuronów, z których zbudowany jest między innymi ludzki mózg. Sieć neuronowa zbudowana jest z neuronów oraz połączeń, które występują między nimi. Połączeniom tym przypisane są wagi, które należą do zbioru liczb rzeczywistych. Neurony rozróżniają także połączenia przychodzące, zwane pobudzeniami, oraz wychodzące. Najczęściej przedstawia się sieć neuronową jak na rysunku 4.1, jako graf kierunkowy ważony, w którym wierzchołki oznaczają neurony natomiast krawędzie to połączenia między nimi. Podstawowym zadaniem neuronu jest sumowanie sygnałów wchodzących, a następnie podaniu wyniku *funkcji aktywacji*. Sygnały wychodzące powstają przez przemnożenie wyniku funkcji aktywacji oraz wagi połączenia. Następnie zostają przetworzone przez kolejne neurony. Iteratywność takiego podejścia przyczyniła się do wyraźnego dzielenia sieci neuronowych na warstwy. Warstwa w sieci neuronowej to zbiór neuronów nie mających połączeń między sobą. Neurony w warstwie przyjmują tylko połączenia przychodzące z warstwy poprzedniej, natomiast połączenia wychodzące łączone są tylko z warstwą następną. Rozmiarem warstwy nazywamy liczbę neuronów, którą dana warstwa posiada. Możemy wyróżnić trzy rodzaje warstw w klasycznej sieci neuronowej:

- warstwa wejściowa
- warstwa ukryta
- warstwa wyjściowa

Warstwa wejściowa nie posiada żadnych połączeń przychodzących. Jej zadaniem jest przekazanie wartości podanych jako wejście do sieci neuronowej do kolejnej warstwy. Z tego powodu rozmiar warstwy wejściowej musi odpowiadać rozmiarowi wektora cech podawanego sieci neuronowej. Warstwy ukryte to warstwy pomiędzy warstwą wejściową i wyjściową. Mają zarówno połączenia przychodzące jak i wychodzące. Liczba tego rodzaju warstw w sieci jest jednym z parametrów, który wpływa na ostateczną skuteczność działania sieci neuronowej. Nazwa tej warstwy pochodzi od braku komunikacji z systemem poza siecią. Warstwa wyjściowa nie posiada połączeń wychodzących. Wynik działania sieci neuronowej pobierany jest bezpośrednio z tej warstwy. Z tego powodu warstwa wyjściowa musi mieć rozmiar taki jak oczekiwany wektor wynikowy.

4.1.1. Sieci rekurencyjne

Klasyczne sieci neuronowe po przeszkoleniu powinny działać bezstanowo. To znaczy, że podając dowolną liczbę razy ten sam wektor wejściowy na wejście sieci neuronowej możemy oczekwać za każdym razem tego samego wektora wyjściowego. W konsekwencji sieci takie mają lepszą skuteczność w problemach nie zależnych znacząco od czasu, takich jak rozpoznanie



Rys. 4.1. Przykład przedstawienia sieci neuronowej
 źródło: <https://impicode.pl/blog/jak-stworzyc-sieć-neuronowa/>

obiektu na obrazie. Natomiast nie sprawdzają się dla problemów kontekstowych, takich jak przewidywanie kolejnego zdania w tekście. Innym przykładem jest problemów przewidywanie *szeregow czasowych*, czyli danych ułożonych w czasie.

Przykład 4.1. Trenujemy sieć neuronową na dwóch zdaniach: *Ala ma kota. Kot ma Ale.* Klasyczna sieć neuronowa po otrzymaniu na wejście **ma** przewidzi z równy prawdopodobieństwem *Ale* i *kota*

Odpowiedią na powyższe problemy są rekurencyjne sieci neuronowe (ang. *Recurrent neural network - RNN*). Sieci te w warstwie wejściowej oprócz standardowego wektora wejściowego przyjmują wektor wyjściowy z poprzedniego wywołania sieci. Pozwala to przekazywać informacje odnośnie poprzednich wejść sieci neuronowej.

4.1.2. LSTM

Problemem klasycznej sieci rekurencyjnej jest szybkie rozmycie stanu, czyli informacji przekazywanej z poprzednich uruchomień sieci. Utrudnia to znajdowanie korelacji między próbками oddalonymi od siebie o znaczne odległości. W celu zaadresowania tego problemu stworzono zmodyfikowaną sieć *RNN* nazwaną *Long Short Term Memory (LSTM)*. Sieci tego rodzaju posiadają dodatkowy stan zwany pamięcią długoterminową. Za modyfikacje tego stanu odpowiadają wyspecjalizowane sieci neuronowe, które definiują jakie wartości zapamiętać, a jakie należy zapomnieć. Z uwagi na czasowe powiązanie danych z giełdy, w naszym projekcie zdecydowaliśmy się użyć sieci *LSTM*

4.2. Trening sieci neuronowej

Treningiem sieci neuronowej nazywamy dostosowanie wag połączeń między neuronami w celu uzyskania największej dokładności. Istnieje wiele sposobów uczenia sieci, jednym z nich jest uczenie nadzorowane. W podejściu tym sieć neuronową trenuje się na przykładach. To znaczy na zbiorze danych, który zawiera pary: wektor wejścia oraz odpowiadający mu oczekiwany wektor wyjścia. Taką pojedynczą parę możemy nazwać próbką. W naszym projekcie wektorem wejścia są dane giełdowe, jak w tabeli 3.1, scalone z danymi sentymentu z tabeli 3.2, natomiast wektorem wyjścia jest pojedyncza cena zamknięcia dnia następnego. Trening sieci możemy podzielić na epoki. Można je interpretować jako pojedyncze iteracje programu przez określoną część przygotowanego zbioru danych. Epoki najczęściej dzielone są na 2 fazy:

- fazę treningu
- fazę walidacji

4.2.1. Faza treningu

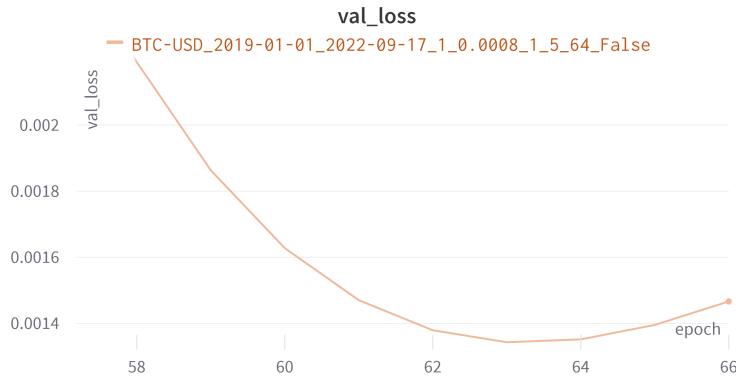
Faza treningu to główna faza każdej epoki. Podczas działania korzysta ze zbioru danych treningowych, czyli wydzielonej części przygotowanego zbioru danych. W tej fazie następuje aktualizacja wag połączeń między neuronami. To czy wagi będą maleć, czy rosnąć oraz jak duża będzie ta zmiana zależy od wyniku funkcji straty. Funkcja straty przyjmuje oczekiwany wektor wyjścia oraz wektor wyjścia będący wynikiem działania sieci neuronowej. Natomiast jako wyjście zwraca skalar oznaczający błąd między podanymi punktami. Istnieje wiele implementacji funkcji straty. Jedną z nich jest błąd średnio-kwadratowy, którego używamy w naszym projekcie. W najprostszej implementacji wagi aktualizowane są dla każdej pary wektorów w zbiorze danych treningowych. Jednak takie podejście nie jest optymalne obliczeniowo. Okazuje się także, że biorąc uśrednioną stratę z kilku próbek podczas aktualizacji wag sieci otrzymujemy lepszą dokładność dla generalnych przypadków nie zawartych w zbiorze treningowym. Z tego powodu próbki przetwarzane są przez program partiami, a aktualizacja wag następuje na podstawie uśrednionej straty próbek w partii. Partie mają zazwyczaj stałą liczbę próbek. Rozmiar partii jest jednym z konfigurowalnych wartości wpływających na ostateczną skuteczność sieci neuronowej. Liczba partii w fazie treningowej zależy od ilości próbek w zbiorze danych treningowych.



Rys. 4.2. Przykład wartości straty w fazie treningu. Pokazana tutaj strata jest uśrednioną stratą z wszystkich partií epoki. Nazwa funkcji zbudowana jest zgodnie z algorytmem z listingu 5.1

4.2.2. Faza walidacji

Trenując sieć neuronową chcemy najczęściej, żeby miała jak największą dokładność dla ogólnego przypadku naszego problemu, a nie dla konkretnych przypadków ze zbioru treningowego. Jednak, z uwagi na to że wagi sieci aktualizowane są w taki sposób żeby zminimalizować stratę w zbiorze treningowym, w pewnej epoce następuje zjawisko przetrenowania. Polega ono na tym, że sieć zaczyna tracić dokładność w przypadkach ogólnych mimo tego, że cały czas zyskuje dokładność dla zbioru treningowego. W celu wyeliminowania tego problemu wprowadzono drugą fazę w epoce. Faza ta uruchamiana jest na koniec epoki po fazie treningu. Korzysta ona z osobnego zbioru danych walidacyjnych. Jej celem jest monitorowanie dokładności sieci dla zbioru nie będącego treningowym. Do tego najczęściej używana jest ta sama funkcja straty, co pozwala łatwo porównywać wyniki. Dzięki obliczaniu po każdej epoce straty na niezależnym zbiorze danych możliwe jest wykorzystanie mechanizmu *wczesnego zatrzymania* (ang. *early stopping*). Mechanizm ten w przypadku pogarszanie się dokładności w wynikach walidacji zatrzyma trening. Efekt ten można zauważyć na wykresie 4.3. W tej fazie nie są aktualizowane wagi sieci.



Rys. 4.3. Przykład wczesnego zatrzymania. Wykres przedstawia średnią wartość straty w fazie walidacji. Nazwa funkcji zbudowanej jest zgodnie z algorytmem z listingu 5.1

4.2.3. Faza Testu

Wytrenowanie skutecznej sieci neuronowej wiąże się z testowaniem różnych konfiguracji i zbiorów danych. W celu łatwego porównania tworzonych modeli warto mieć przygotowany zbiór danych testowych, na którym będziemy mogli sprawdzić dokładność wytrenowanego modelu. Takie czynności nazywamy fazą testową. Następuje ona po wszystkich epokach treningu. Dla ułatwienia porównywania powinna zwracać skalar dokładności bądź straty. Podczas tej fazy nie są aktualizowane wagi sieci.

4.3. PyTorch Lightning

Gdy wybraliśmy architekturę sieci *LSTM*, przystąpiliśmy do wyboru sposobu jej implementacji. Jedną z opcji było stworzenie architektury od podstaw w wybranym języku programowania. Jednak takie podejście wymaga dużego wkładu czasowego i jest poza zakresem naszej pracy. Dużo częstszym podejściem do implementacji jest skorzystanie z jednego z dostępnych na rynku frameworków. Frameworki te udostępniają API, które ułatwia tworzenie i trening modeli sztucznej inteligencji. Najpopularniejsze frameworki są dedykowane dla języka programowania *Python*, gdyż oferuje on prostą składnię i wiele bibliotek do operacji na macierzach jak przykładowo *numpy*. W początkowej fazie projektu zdecydowaliśmy się skorzystać z frameworka *PyTorch*. Jest on jednym z najczęściej wybieranych frameworków, ponieważ posiada szczegółową dokumentację oraz dużą bazę poradników. Wybraliśmy go także ze względu na możliwość kompleksowej konfiguracji i modyfikacji sieci. *PyTorch* jest zaimplementowany w języku *C++*, który według porównań działa o rząd wielkości szybciej niż *Python*. To wraz z wykorzystaniem procesora graficznego przy obliczeniach sprawia, że *PyTorch* jest szybki.

W trakcie rozwijania projektu natrafiliśmy na kilka problemów z użyciem *PyTorch*. W celu zaimplementowania naszej sieci musieliśmy zaimplementować wiele linii powtarzalnego kodu, co utrudniało nam szukanie błędów. Brakowało nam także funkcji umożliwiających zapis logów z procesu treningu w sposób ustukturalizowany. Kolejny problem dotyczył przedstawiania wyników działania sieci. Dlatego zdecydowaliśmy się użyć rozszerzenia *PyTorch* o nazwie *Pytorch Lightning*. Jest on w pełni kompatybilny z oryginałem przez co migracja naszego projektu była naturalna. Główną zaletą tego frameworka jest wymuszenie ścisłego podziału logiki kodu, co zwiększa znacząco jego czytelność. Pozwala on także na wykorzystanie zaimplementowanych

już mechanizmów jak przykładowo *wczesne zatrzymanie*, czy integracja z serwisami które wspomagają monitorowanie treningu.

4.4. Konfiguracja sieci i znaczenie parametrów

Aby stworzyć najbardziej skuteczną sieć neuronową, wymagany jest nie tylko odpowiednio dobrany i znormalizowany zbiór danych. Należy również dobrać odpowiednie parametry treningu. Dobór parametrów w celu znalezienia optymalnej sieci jest zadaniem nietrywialnym. Jest tak, gdyż optymalne parametry zależą od konkretnego problemu i zbioru danych. Pomocą przy ustalaniu parametrów jest poznanie ich znaczenia. W naszej pracy skupiliśmy się na poznaniu najczęściej wykorzystywanych parametrów przy szkoleniu sieci neuronowej w *frameworku PyTorch lightning*:

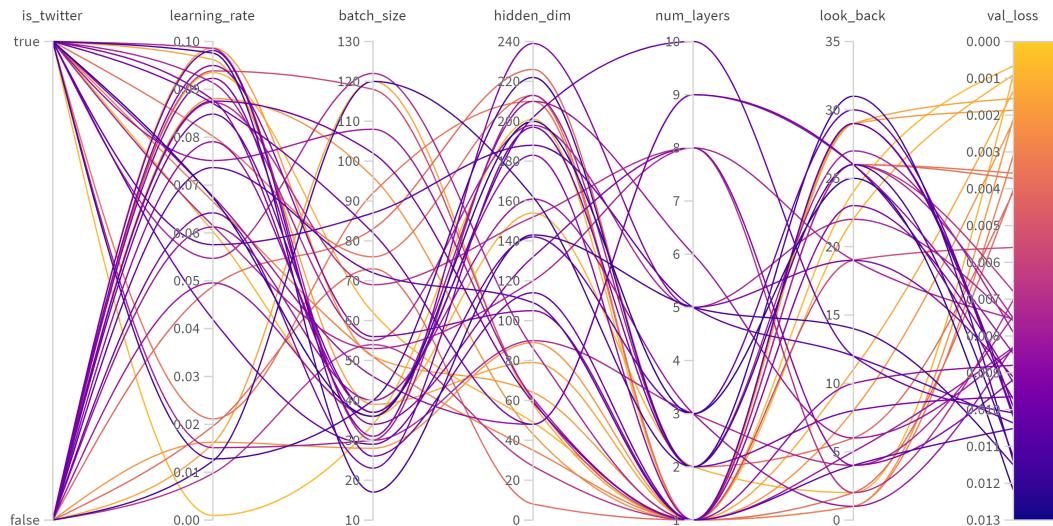
- liczba warstw
- rozmiar warstwy ukrytej
- długość sekwencji
- rozmiar partii (ang. *batch size*)
- wskaźnik uczenia (ang. *learning rate*)

Liczba warstw odnosi się do liczby warstw ukrytych. Na przykład, ustawienie liczby warstw na 2 oznacza ułożenie dwóch sieci LSTM razem. Druga sieć LSTM przyjmuje jako wektor wejściowy wektor wyjścia pierwszej sieci LSTM. Rozmiar warstwy ukrytej wpływa znacząco na trening, lecz zwiększenie jego wartości nie przekłada się wprost proporcjonalnie na ostateczną skuteczność sieci. Długość sekwencji jest parametrem specyficznym dla sieci z rodziny *RNN*. Określa ona ile próbek zostanie podanych na sieć neuronową. Podczas obliczania funkcji straty pod uwagę brany jest tylko wektor wyjścia z ostatniej próbki w sekwencji. W przypadku sieci LSTM reszta próbek służy do wypełnienia stanu pamięci krótko i dugo terminowej. Rozmiar partii, w przypadku sieci rekurencyjnych mówi ile sekwencji próbek ma się mieścić w danej partii. *Learning rate* wpływa na to, jak model uczenia maszynowego korzysta z informacji zwrotnych z funkcji straty przy aktualizacji wag. Jeśli *learning rate* jest zbyt mały, model będzie się uczył wolno, ponieważ każda korekta będzie niewielka. Jeśli jest zbyt duży, model może "przeskoczyć" przez optymalne rozwiązanie i dostarczyć gorsze wyniki.

4.4.1. Optymalizacja konfiguracji

W celu odnalezienia najlepszej konfiguracji sieci skorzystaliśmy z narzędzia *Wandb sweep*. Narzędzie to zarządza dostarczaniem wektora parametrów do treningu zgodnie z ustaloną przez programistę strategią. Na strategię składa się metoda dostarczania parametrów oraz zakres możliwych parametrów. Dostępne są trzy metody dostarczania parametrów: *grid* podający każdą możliwą kombinację parametrów, *random* podający losowy zbiór parametrów oraz *bayes*, w którym działanie narzędzia opiera się na modelu probabilistycznym i stara się wybrać jak najlepszy zbiór parametrów. W naszym projekcie wybraliśmy metodę *random* z zakresem parametrów przedstawionych na wykresie 4.4

Analizując treningi, z wykresu 4.4, mające najmniejszą stratę w ostatniej fazie walidacji, wybraliśmy konfiguracje użytą do porównywania modeli szkolonych na różnych zbiorach danych. Konfigurację tę przedstawiliśmy w tabeli 4.1



Rys. 4.4. Wynik działania *wandb sweep* każda linia przedstawia pojedyncze trenowanie modelu. powyżej przedstawiono 40 najlepszych treningów z 571 przeprowadzonych

Tabela 4.1. Wybrana konfiguracja sieci neuronowej na podstawie najlepszego treningu z wykresu 4.4

Nazwa parametru	wartość
liczba warstw	1
rozmiar warstwy ukrytej	200
długość sekwencji	24
rozmiar partii	64
wskaźnik uczenia	0.0008

4.5. Wstępne przetworzenie danych

W sieciach neuronowych wagi zwykle mają wartość zbliżoną do jedności [6]. Dzięki temu większość algorytmów aktualizujących wagi sieci zgodnie z funkcją straty lepiej radzi sobie z małymi wartościami. Taka sytuacja pozwala na efektywniejsze przetwarzanie danych w sieci neuronowej. W przygotowanym zbiorze mogą także występować próbki, które nie posiadają wszystkich oczekiwanych przez nas cech. Zbiór danych trzeba także podzielić na podzbior treningowy, walidacyjny oraz testowy. Dlatego przed treningiem musielibyśmy podać nasz zbiór danych wstępнемu przetworzeniu. Proces ten można podzielić na 3 kroki:

1. uzupełnienie brakujących cech
2. podział zbioru na podzbiory
3. normalizacja danych

4.5.1. Uzupełnienie brakujących cech

Aby dać użytkownikom naszej aplikacji, a pośrednio użytkownikom naszej sieci, możliwość przewidywania ceny aktywa w dowolnym dniu, musielibyśmy dysponować zbiorem danych, który pokrywa każdy dzień w określonym z góry zakresie czasu. Dane odnośnie sentymentu nie były problematyczne, gdyż są one generowane w sposób ciągły. Natomiast w danych giełdowych brakuje próbek z dni, w których giełda nie pracuje, to znaczy świąt i weekendów. Brak tych danych w przypadku szkolenia sieci powodowałby pogorszenie jej dokładności. Co więcej, trening sieci równocześnie z danymi na temat sentymentu byłby niemożliwy. Powodem tego jest występowanie brakujących cech w wektorze wejścia stworzonym z połączenia danych giełdowych i sentymentu.

Istnieje wiele sposobów na wypełnienie brakujących cech w danych, które są szeregiem czasowymi. Przykładowo, można interpolować brakującą wartość na podstawie sąsiadujących próbek. Biorąc pod uwagę specyfikę giełdy zdecydowaliśmy się na metodę *fill forward*. Polega ona na wypełnianiu brakujących cech w wektorze, bądź całych wektorów, przy użyciu próbki poprzedzającej. W naszym przypadku, gdy nie mamy danych na dany dzień przepisujemy, je z dnia poprzedniego. Jest to prosty, ale skuteczny sposób wypełniania brakujących cech.

4.5.2. Podział zbioru na podzbiory

Zgodnie z opisem treningu sieci neuronowej otrzymany zbiór danych musimy podzielić na 3 podzbiory. Nie ma z góry określonych proporcji wynikowych podzbiorów, dobrą zasadą jest, żeby zbiór treningowy posiadał najwięcej próbek. W naszym projekcie wybraliśmy następujące proporcje odpowiednio dla zbioru treningowego, walidacyjnego i testowego odpowiednio: 8 : 1 : 1. Pozwala to zachować wiarygodność wyników fazy walidacji i testu równocześnie nie ograniczając znacząco zbioru danych w fazie treningu. W zbiorach danych nie związanych czasowo najczęściej wybór próbek do danego zbioru jest losowany. Dzięki temu uzyskujemy równomierny rozkład cech w poszczególnych podzbiorach danych. Jednak, jako że nasze dane są szeregiem czasowym, a dane w podzbiorach powinny być ciągle czasowo, musieliśmy wybrać inną metodę podziału. Obecnie w naszym programie dla zbioru danych uszeregowanych od najpóźniejszej próbki do najwcześniejjszej, pierwsze 80% próbek zaliczamy do zbioru treningowego. Następne 10% do zbioru walidacyjnego, a ostatnie 10% do zbioru testowego.

4.5.3. Normalizacja danych

Normalizacja to dostosowanie wartości mierzonych w różnych skalach do jednej skali w celu ułatwienia porównywania tych wartości. Normalizacja danych nie tylko zwiększa stabilność treningu sieci. Wpływa także na jej dokładność eliminując sztuczną faworyzacje cech mających większą wartość. W naszym projekcie wartość cechy *Volume* była kilka rzędów wielkości większa niż cena akcji co powodowało spadek dokładności sieci. Istnieje wiele sposobów normalizacji. W naszym projekcie zastanawialiśmy się nad użyciem dwóch rozwiązań: standaryzacji oraz skalowania min-max. Standaryzacja oddaje każdą kolumnę w zbiorze danych równaniu z definicji 4.2. Takie podejście jest mało wrażliwe na elementy odstające. Niestety otrzymany przedział wartości zależy od zbioru danych przez co porównywanie modeli wytrenowanych dla różnych aktywów jest problematyczne.

Definicja 4.2. Standaryzacja Z

$$Z = \frac{x - \mu}{\sigma}$$

gdzie:

- x – zmienna niestandardyzowana,
- μ – średnia z populacji,
- σ – odchylenie standardowe populacji.

Skalowanie *min-max* oddaje każdą kolumnę w zbiorze danych równaniu 4.3. Celem skalowania *min-max* jest przeskalowanie wszystkich wartości w danej kolumnie do zadanego zakresu najczęściej $< 0,1 >$. Daje to możliwość łatwego porównywania działania modeli niezależnie od rozpiętości wartości w zbiorze danych. Jednakże występowanie elementów odstających w podanym sposobie może zmniejszyć znacząco rozdzielcość otrzymanego zbioru. Biorąc pod uwagę cel porównywania modeli trenowanych na aktywach różnych firm oraz zbiór danych, w którym nie

występują elementy odstające, zdecydowaliśmy się użyć sposobu skalowania *min-max*. Ważną uwagą jest konieczność dopasowania wartości skalowania tylko na zbiorze testowym, natomiast na reszcie zbiorów wykonać tylko skalowanie. Zapobiega to sztucznemu zawyżaniu wyników w fazie walidacji i testu.

Definicja 4.3. Skalowanie *min-max*

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)}$$

gdzie:

- x – zmienna niestandardyzowana,
- $\min(x)$ - minimalna wartość w zbiorze
- $\max(x)$ - maksymalna wartość w zbiorze

4.6. Monitorowanie i analiza wyników

Monitorowanie obecnego stanu treningu jest ważne z wielu powodów. Dzięki monitorowaniu możemy porównać przebieg szkolenia dla modeli trenowanych z różnymi parametrami i zbiorami danych. Pozwala nam to także w łatwy sposób korzystać z funkcji takich jak wcześnie zatrzymanie. W początkowych fazach projektu monitorowaliśmy średnią *stratę* w danej epoce dla fazy treningu i walidacji oraz stratę w fazie testu, zebrane dane analizowaliśmy na wykresach stworzonych przy użyciu biblioteki *matplotlib*. Takie podejście nie pozwalało łatwo dzielić się wynikami w zespole, brakowało mu także możliwości porównywania.

4.6.1. Wandb

Rozwiązaniem powyższych problemów jest serwis *Wandb*. Serwis ten oferuje aplikację internetową dedykowaną do porównywania modeli. Dzięki integracji z *Pytorch Lightning* dane przesyłane są w trakcie treningu do serwisu, co umożliwia śledzenie przebiegu treningu w czasie rzeczywistym. W przypadku długich treningów jest to bardzo pożąданie. *Wandb* pozwala przesyłać dane z wielu komputerów do jednego projektu, który zawiera w sobie wszystkie przeprowadzone treningi. Dzięki temu rozwiązaliśmy problem łączenia danych tworzonych na osobnych maszynach. Jedyną wadą tego rozwiązania jest wydłużenie czasu treningu o około 5 sekund.

4.6.2. Skuteczność

Podczas rozwijania projektu zauważliśmy potrzebę mierzenia wartości, która w bardziej semantyczny sposób pozwalałaby nam określić dokładność naszej sieci. Dotychczas używana strata w fazie testowej była na poziomie 10^{-3} , co trudno było porównać manualnie. Stwierdziliśmy także, że dla użytkownika najważniejszą informacją jest to, w jakim kierunku zmieni się cena akcji. Dlatego zdecydowaliśmy się wprowadzić współczynnik dokładności obliczany zgodnie z algorytmem 2. Współczynnik ten określa jaki procent przewidywania wskazał właściwy trend zmiany ceny. Pokazuje to w przybliżony sposób użyteczność naszej sieci. Jednak podejście nie uwzględnia straty między przewidywaną, a oczekiwana ceną aktywa, dlatego korzystając z tego współczynnika przy porównywaniu należy mieć na uwadze także stratę.

Algorithm 2 Obliczanie skuteczności sieci

```
1: function CZY_SKUTECZNY(cena_z_dnia_predykci, cena_przewidziana, cena_oczekiwana)
2:   if (cena_z_dnia_predykci  $\leq$  cena_przewidziana  $\&$  cena_z_dnia_predykci  $\leq$  cena_oczekiwana)  $(cena_z_dnia_predykci \geq cena_przewidziana \& cena_z_dnia_predykci \geq cena_oczekiwana) then
3:     return Prawda
4:   end if
5:   return Fałsz
6: end function$ 
```

5. WARSTWA LOGIKI BIZNESOWEJ

W początkowych fazach projektu zarządzaliśmy konfiguracją sieci bezpośrednio poprzez modyfikacje wartości w skrypcie. Sposób ten był wystarczający, gdy przeprowadzaliśmy pierwsze próby działania naszej sieci. Jednak przez brak elastyczności tego rozwiązania, zdecydowaliśmy się stworzyć serwis pozwalający konfigurować zachowanie treningu przy użyciu protokołu *HTTP*. W trakcie rozwijania projektu zwiększałyśmy jego możliwości i obecnie serwis ten jest odpowiedzialny za całą logikę związaną w siecią neuronową. Serwis ten nazwaliśmy *Backend*. Do implementacji serwisu wykorzystałyśmy język programowania *Python*. Nasza decyzja wynikała z chęci łatwej integracji z zaimplementowanym już przez nas treningiem sieci neuronowej. W celu umożliwienia odbierania przez nasz serwis zapytań *HTTP* użyliśmy *frameworka Flask*.

5.1. *Endpointy*

Serwis *Backend* udostępnia swoje funkcjonalności aplikacji internetowej poprzez cztery *endpointy*:

- */train* typu *POST*
- */predict* typu *POST*
- */models* typu *GET*
- */assets* typu *GET*

5.1.1. */train*

Głównym *endpointem* naszego serwisu jest ten odpowiedzialny za trenowanie sieci. Zapytanie pod ten *endpoint* można parametryzować zgodnie z tabelą 5.1

Tabela 5.1. Dostępne parametry dla zapytania */train*

Nazwa parametru	Format danych	Wartość domyślna
token	str	"AMZN"
start_date	datetime.date	2018-1-1
end_date	datetime.date	dzisiejsza data
number_of_layers	int	1
hidden_dim	int	32
learning_rate	float	0.0008
look_back	int	5
batch_size	int	64
is_twitter	bool	TRUE

Działanie

W pierwszej kolejności program przetwarza ciało zapytania, będące w formacie *JSON*, i odwzorowuje je na stworzoną przez nas klasę *ModelParams*, której struktura odpowiada tabeli 5.1. Procesujemy także wstępnie podany string oznaczający datę w celu usunięcia zbędnych informacji jak godziny i minuty. Następnie wywołujemy funkcję odpowiedzialną za trening. W funkcji tej na początku ustawiamy *seed* wszystkich bibliotek zapewniających randomizację na 0, ponieważ *PyTorch Lightning* używa algorytmów niedeterministycznych. Powoduje to łatwiejsze porównywanie modeli między sobą, lecz może nie prowadzić do najlepszej optymalizacji sieci. Następnie tworzymy klasę *DataModule* odpowiedzialną za przygotowanie danych do treningu. W przypadku

szkolenia z danymi z *Twittera*, klasa ta poszukuje pliku CSV z nazwą *token*.csv* w folderze */dataset/sentiment*. Plik ten powinien zawierać dane na temat sentymentu w formacie zgodnym z tabelą 3.2. Natomiast dane dotyczące giełdy pobierane są przy użyciu biblioteki *yfinance*. W kolejnym kroku dane są scalane do jednej tabeli, a następnie poddane wstępнемu procesowaniu zgodnie z podanymi w sekcji 4.5 krokami. Następnie tworzymy instancje klasy modelu sieci LSTM przy pomocy *ModelParams*. Parametr *input_size*, którego nie podajemy w zapytaniu, wnioskowany jest z liczby kolumn w przygotowanych przez *DataModule* danych. Kolejnym krokiem jest stworzenie instancji klasy *pytorch_lightning.Trainer*. Odpowiada ona za przeprowadzenie treningu naszej sieci. Sposób treningu można modyfikować przy użyciu parametrów. Szczególnie przydatna jest możliwość podania listy klas dziedziczących po klasie *callback*, które pozwalają rozszerzyć działanie treningu o dodatkowe akcje. Klasy te posiadają funkcje wywoływane w konkretnych miejscach treningu, jak na przykład koniec fazy walidacji. W naszym projekcie przekazujemy po jednej instancji z trzech takich klas:

- *EarlyStopping*
- *ModelCheckpoint*
- *MetricsCallback*

EarlyStopping odpowiada za mechanizm wczesnego zatrzymywania treningu sieci. Monitoruje wartość straty podczas fazy walidacji. Parametr *patience* tej klasy, mówi przez ile epok musi się utrzymać pogarszająca tendencja monitorowanej wartości, by przerwać trening. Nadaliśmy mu wartość 3, co z naszych obserwacji dawało najlepsze wyniki.

ModelCheckpoint to klasa dostarczana przez framework *PyTorch Lightning*, pozwalająca kontrolować zapisywanie modeli. W naszym projekcie skonfigurowaliśmy instancje tej klasy tak, że modele zapisywane są w folderze */models/nazwa_konfiguracji*, gdzie *nazwa_konfiguracji* to tworzony na postawie *ModelParams* napis. Sposób tworzenia przedstawiony jest w listingu 5.1.

```
1 def create_name(p: ModelParams) -> str:
2     return f"{p.token}_{p.start_date}_{p.end_date}_{p.number_of_layers}_{p.learning_rate}"
3     }_{p.hidden_dim}_{p.look_back}_{p.batch_size}_{p.is_twitter}"
```

Listing 5.1: Tworzenie nazwy konfiguracji

Natomiast nazwa pliku powstaje z połączenia straty w fazie walidacji oraz obecnej epoki. Ostatecznie relatywna ścieżka modelu wygląda jak w przykładzie 5.1

Przykład 5.1. Dla zapytania z parametrami domyślnymi nazwa modelu w epoce 123:

models/AMZN_2018-01-01_2022-10-13_1_0.0008_1_5_64_True/val_loss=0.0001-epoch=123.ckpt

PyTorch Lightning zapisuje stan modelu z ostatniej epoki, lecz podając wartość do monitorowania możemy wymusić zapisywania tylko modelu z najlepszą wartością. W naszym przypadku monitorujemy wartość straty w fazie walidacji. Wraz z modelem zapisywane są wszystkie parametry potrzebne do przewidzenia przyszłych cen aktyw, takie jak wartości funkcji odpowiedzialnych za skalowanie *min-max* użytych przy wstępny przetwarzaniu danych.

MetricsCallback to stworzona przez nas klasa odpowiedzialna za zapisywanie stanu treningu pod koniec każdej epoki.

Następnie przy pomocy stworzonej instancji klasy *Trainer* przeprowadzamy trening oraz uruchamiamy fazę testową. Ostatecznie konwertujemy dane zebrane przez instancje klasy *MetricsCallback* do formatu *JSON* i wysyłamy je w ciele odpowiedzi na żądanie do klienta.

5.1.2. /predict

W celu umożliwienia przewidywania z wykorzystaniem wyszkolonych przez nas modeli, stworzyliśmy *endpoint* pozwalający przewidzieć cenę zamknięcia danego aktywa. Cena aktywa przewidywana jest na dzień następny po podanej dacie. Jako wejście do sieci neuronowej podawane są dane z x dni poprzedzających podany dzień. Zapytanie to można parametryzować zgodnie z tabelą 5.2

Tabela 5.2. Dostępne parametry dla zapytania /predict

Nazwa parametru	Format danych	Wartość domyślna
model_name	str	-
date	datetime.date	dzisiejszy dzień
lookback	Optional[int]	-

Działanie

Pierwszym krokiem jest przetworzenie ciała zapytania na stworzoną przez nas klasę *PredictParams*, następnie wywołujemy funkcję *predict* której działanie możemy podzielić na trzy kroki:

1. załadowanie najlepszego modelu
2. pobranie i wstępne przetworzenie danych
3. predykcja i skalowanie wyniku

Parametr *model_name* z tabeli 5.2 oznacza nazwę konfiguracji modelu według konwencji z listingu 5.1. Nazwa ta zgodnie z opisem *endpointu* /predict wskazuje na folder zawierający modele z daną konfiguracją. W przypadku przeprowadzenia kilku treningów z tą samą konfiguracją w danym katalogu będzie zapisany więcej niż jeden model. Dlatego na potrzeby predykcji zdecydowaliśmy się wybierać model z najlepszą wartością straty w fazie Walidacji ostatniej epoki. Wartości tą wyciągamy z nazwy zapisanego modelu. Po wybraniu najlepszego modelu pobieramy go z pliku zapisu. Wraz z nim modelem pobierane są także informacje potrzebne do dalszego przeprowadzenia predykcji.

Następnym krokiem jest pobranie potrzebnych danych. Liczba wierzy do pobrania określana jest na podstawie parametru *lookback*, lub w przypadku jego braku na podstawie wartości *lookbacku* na której trenowany był model. Dane akcyjne pobierane są na dany dzień przy pomocy biblioteki *yfinance*, natomiast w przypadku sieci trenowanych z użyciem twittera, dane na temat sentymentu pobierane są bezpośrednio z naszego serwisu. Użycie tutaj dynamicznego pobierania sentymentu pozwala nam uniezależnić predykowanie od przygotowanych już zbiorów danych, które mogą nie zawierać żądanego przez klienta okresu czasu. Wydłuża to jednak znaczaco czas predykcji, gdyż pobranie sentymentu dla 5 dni wiąże się z czasem oczekiwania około 20 sekund. Pobrane dane podajemy następnie skalowaniu przy użyciu tego samego skalera co przy przetwarzaniu danych treningowych.

w ostatnim kroku podajemy na wejście sieci przygotowane dane. Otrzymany wynik poddajemy odwrotnej transformacji przy użyciu scalera zapisanego wraz z modelem a następnie zwracam go w ciele odpowiedzi.

5.1.3. Endpointy informacyjne

Podczas tworzenia naszego projektu zauważaliśmy potrzebę dostępu do informacji o zasobach dostępnych dla *Backendu*. W szczególności jakie zbiory danych o sentymencie są gotowe do wykorzystania oraz modele z jakimi parametrami zostały już wytrenowane. W celu zaadreso-

wania tych problemów stworzyliśmy dwa *endpointy* nie przyjmujące parametrów. Pierwszy */models* zwraca listę nazw wszystkich folderów w katalogu *./models/*. Drugi */assets* przeszukuje folder *./dataset/sentiment/* i na podstawie nazw zawartych tam plików zwraca listę słowników z parametrami modelu. Słowniki te mają strukturę:

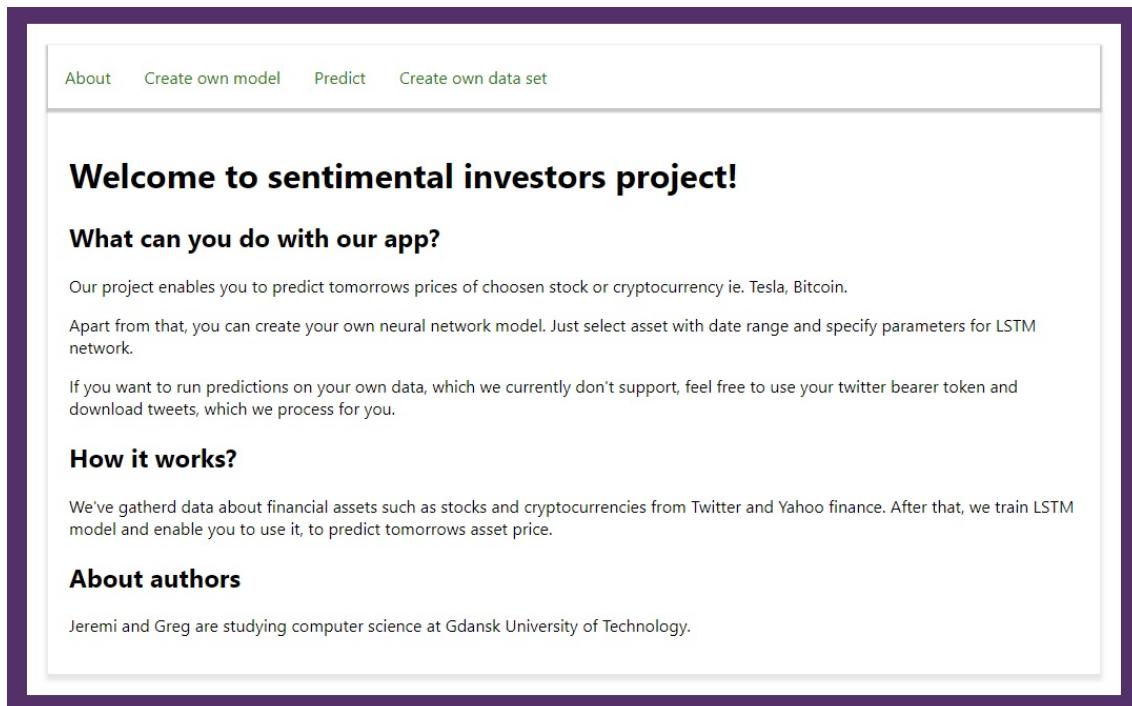
```
1 {  
2     "token":      token,  
3     "name":       pelna nazwa firmy brana z przygotowanego przez nas slownika ,  
4     "start_date": data pierwszej probki ,  
5     "end_date":   data ostatniej probki  
6 }
```

Listing 5.2: lista słowników z parametrami modelu

Dają one informacje o dostępnych zbiorach danych sentymentu.

6. APLIKACJA UŻYTKOWNIKA

W celu pokazania możliwość i kluczowych kroków naszego projektu, stworzyliśmy przeglądarkową aplikację użytkownika. Składa się ona ze strony głównej oraz czterech podstron. Podstrona *About* opisuje w skrócie projekt (nieformalnie nazwaliśmy go *Sentimental investors*). Podstrona *Create own model* pozwala na stworzenie własnego, sparametryzowanego modelu sieci neuronowej w dwóch wersjach. W kolejnej zakładce możemy stworzyć własny zbiór danych z *Twittera* i zarazem wygenerować plik zawierającego wynikowy sentyment przetworzonych tweetów. Ostatnia, najbardziej oczekiwana podstrona, oferuje przewidywanie jutrzeszej ceny danego aktywa finansowego.



Rys. 6.1. Strona opisująca projekt

6.1. Wybrana technologia

Aplikację napisaliśmy przy użyciu biblioteki *ReactJs*. Wystylizowaliśmy ją przy pomocy stylów *Css*.

6.1.1. ReactJs

ReactJs to biblioteka języka *JavaScript*. Pozwala na tworzenie zaawansowanego interfejsu użytkownika. Swoje działanie zasadza na wykorzystaniu tzw. komponentów, czyli fragmentów kodu, które mogą być wielokrotnie wykorzystywane (podejście *reusable*). Komponenty mogą być klasami i wtedy rozszerzają działanie klasy *Component*. Drugim możliwym podejściem jest tworzenie komponentów funkcyjnych. Do najważniejszych różnic między nimi należą:

- komponent klasowy odpowiada za logikę i przechowuje stan, komponent funkcyjny przyjmuje dane jako parametr i renderuje je

- komponent klasowy musi mieć metodę `render()`
- w komponencie funkcyjnym nie można wykorzystywać metod cyklu życia (*React lifecycle methods*) takich jak `componentDidMount()`

6.1.2. Fragmenty kodu

W dalszej części prezentujemy wybrane fragmenty naszego kodu obrazujące obydwa rodzaje komponentów.

Przykład komponentu funkcyjnego, który renderuje pasek z nawigacją w naszym projekcie w listingu 7.4

Przykład komponentu klasowego, który obsługuje logikę wyboru daty. Co ciekawe, korzysta on z komponentu *DatePicker* zdefiniowanego w module *react-date-picker* w listingu 7.5.

6.1.3. Najważniejsze wykorzystane zależności

Do najważniejszych wykorzystanych w projekcie zależności należą:

- *react-select*: biblioteka zawierającą estetyczne i konfigurowalne pola *input* typu *select*.
- *recharts*: biblioteka służąca do rysowania wykresów.
- *http-proxy-middleware*: biblioteka pomagająca przekierowywać żądania *HTTP*. Wykorzystamy ją do skomunikowania aplikacji użytkownika z *backendem*.

6.2. Przykład użycia

6.2.1. Generowanie sparametryzowanego modelu sieci neuronowej

W ramach tej funkcjonalności użytkownik na początku wybiera aktywo z dostępnych. Aktywa dostępne to takie, dla których wcześniej przygotowaliśmy dane *Twittera*. Obecnie użytkownik może wybierać spośród kilkunastu aktywów, którymi są akcję lub kryptowaluty.

Po wybraniu aktywa, aplikacja automatycznie ustala najszerzy możliwy zakres dat (im większa rozpiętość zakresu tym więcej danych podanych na wejście sieci).

Następnie użytkownik może podać konkretne parametry sieci: liczbę warstw, współczynnik uczenia, liczbę ukrytych cech, rozmiar serii danych (*lookback*), liczba serii, po której następuje propagacja wsteczna (*batch size*). Następnie użytkownik wysyła formularz z podanymi danymi i serwer uruchamia proces trenowania modeli. Pierwszy model jest trenowany tylko z użyciem danych z giełdy, drugi z kolei jest trenowany z użyciem danych z giełdy i z *Twittera*. W przypadku podania nieprawidłowych danych, użytkownik jest informowany o błędzie.

About Create own model Predict Create own data set

Choose request params

Choose asset

Choose date

From: 25.11.2022

To: 25.11.2022

Choose NN params

Choose nr of layers

Choose learning rate

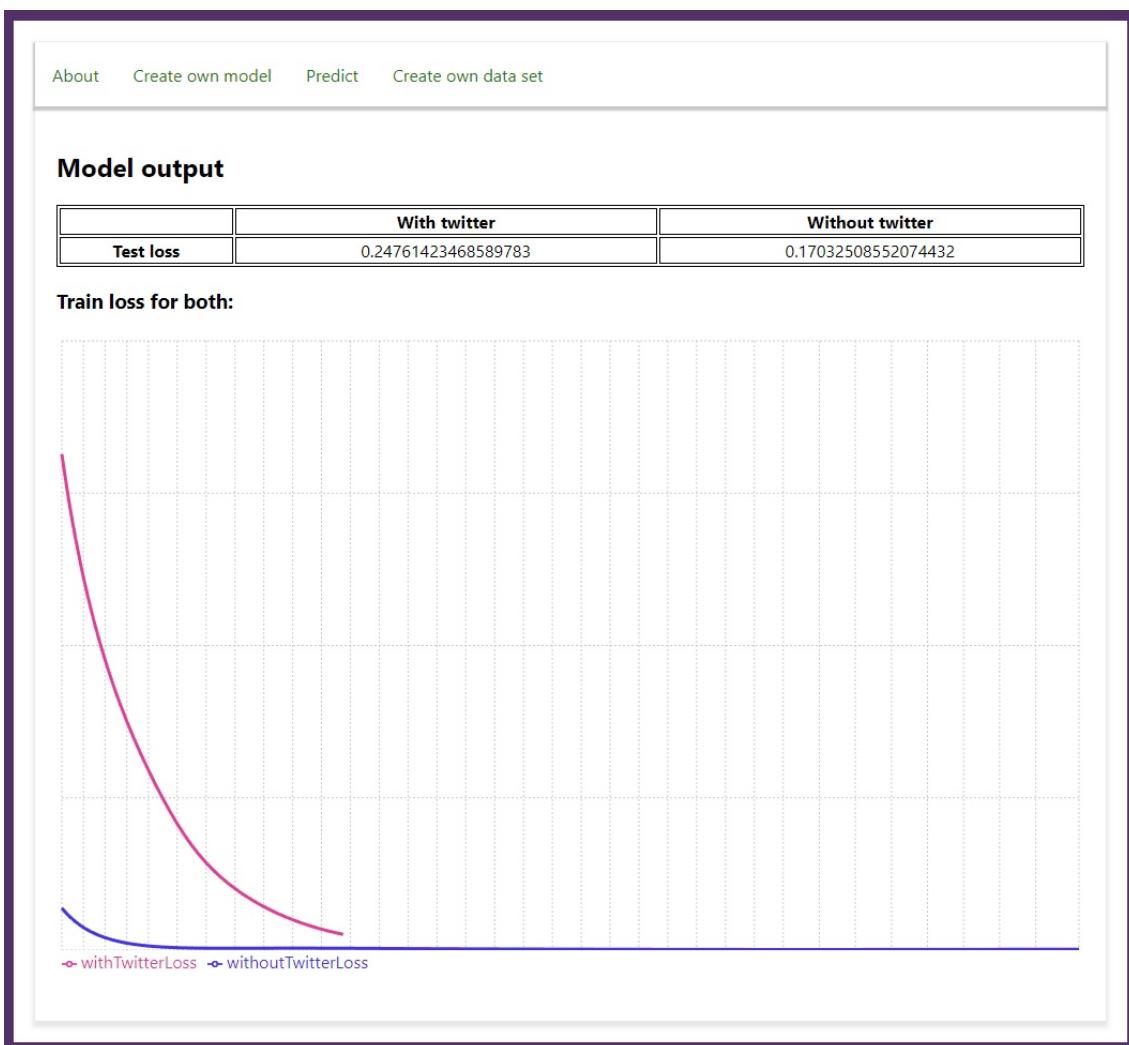
Choose nr hidden dimensions

Choose lookBack

Choose batch size

Rys. 6.2. Strona z formularzem do tworzenia sparametryzowanego modelu

Po zakończonym treningu użytkownik jest przekierowywany do podstrony z wynikami uczenia modeli (*train loss*) oraz wykresami, które przedstawiają wynik działania modeli (*test loss*).



Rys. 6.3. Strona z wynikami tworzenia sparametryzowanego modelu

6.2.2. Tworzenie zbioru danych z Twittera

Wykorzystując tę funkcjonalność, użytkownik może stworzyć własny zbiór składający się z tweetów zawierających wybraną frazę w zadanym okresie. W tym celu użytkownik wyszukuje interesujące go aktywo, zakres dat, frazę, którą mają zawierać pobierane tweety, a następnie podaje swój *bearer token* - służący do uwierzytelnienia się w *Twitter API* - oraz email i wysyła formularz.

Następuje uruchomienie procesu, który pobiera tweety. Gdy wszystkie tweety spełniające kryteria są pobrane, serwer tworzy plik zawierający sentyment oraz statystki przetworzonych tweetów. Po wejściu w podstronę z tworzeniem własnego modelu, w liście dostępnych aktywów finansowych widać symbol (*ticker*) firmy, dla której utworzono zbiór danych.

About Create own model Predict Create own data set

Choose asset

From: 25.11.2022

To: 25.11.2022

Type in query to scrap chosen tweets

Type in your twitter bearer token

Type in your email

Prześlij

Rys. 6.4. Strona służąca do tworzenia własnego zbioru danych

6.2.3. Przewidywanie ceny danego aktywa finansowego na kolejny dzień

W ramach tej funkcjonalności użytkownik może wybrać wytrenowany model z dostępnych i przewidzieć cenę danego aktywa na kolejny dzień.

The screenshot shows a web-based application for model selection and prediction. At the top, there is a navigation bar with links: 'About', 'Create own model', 'Predict', and 'Create own data set'. Below this, a section titled 'Choose model' contains a list of six model configurations, each represented by a button-like box:

- token: INTC, start time: 2014-01-01, end time: 2022-10-10, hidden dim.: 1, lookback: 5, is twitter: true
- token: INTC, start time: 2014-01-01, end time: 2022-10-10, hidden dim.: 1, lookback: 5, is twitter: false
- token: INTC, start time: 2014-01-01, end time: 2022-10-10, hidden dim.: 3, lookback: 5, is twitter: false
- token: INTC, start time: 2020-11-13, end time: 2022-10-10, hidden dim.: 1, lookback: 5, is twitter: false
- token: INTC, start time: 2014-01-01, end time: 2022-10-10, hidden dim.: 3, lookback: 5, is twitter: true
- token: INTC, start time: 2020-11-13, end time: 2022-10-10, hidden dim.: 1, lookback: 5, is twitter: true

At the bottom of this section is a green 'Predict' button.

Rys. 6.5. Strona służąca do przewidywania

Po zakończeniu działania modelu, użytkownik jest przekierowywany do strony z wynikiem przewidywania, gdzie może zobaczyć jutrzejszą cenę aktywa dla, którego uruchomił proces.

The screenshot shows a web-based application displaying the results of a prediction. At the top, there is a navigation bar with links: 'About', 'Create own model', 'Predict', and 'Create own data set'. Below this, the text 'Model prediction for INTC for 26.11.2022 (tomorrow)' is displayed, followed by the predicted value '29.76\$'.

Rys. 6.6. Strona zawierająca wyniki przewidywania

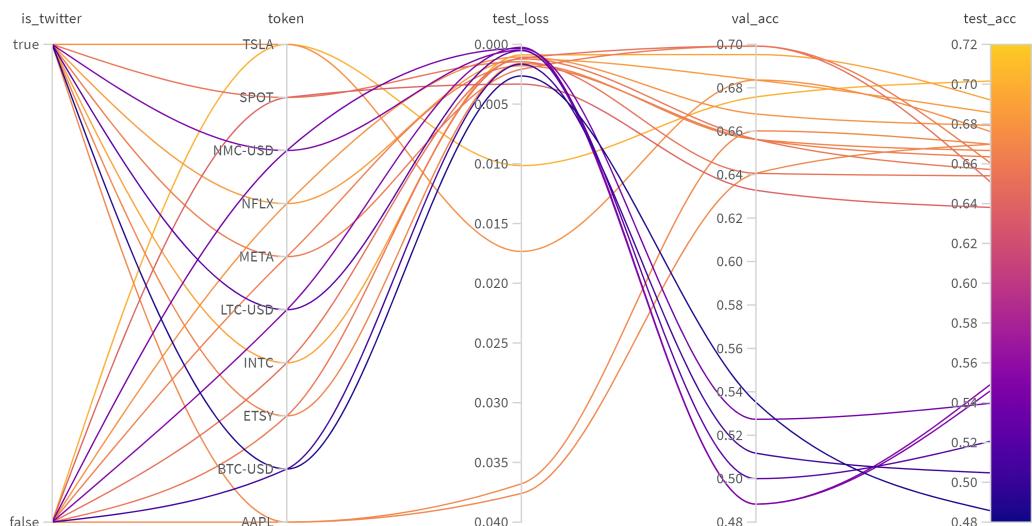
7. PODSUMOWANIE

W ramach naszego projektu próbowaliśmy odpowiedzieć na pytanie, czy dobór podawanych na wejście danych wpływa na wyższą skuteczność przewidywania przyszłej ceny wybranego aktywa finansowego przez sieć neuronową. Po przeprowadzeniu serii testów, zaobserwowaliśmy, że wykorzystanie danych z sentymentem nie wpływa znacząco na skuteczność przewidywania. Stworzyliśmy również przeglądarkową aplikację użytkownika, która pozwala na stworzenie zbioru treningowego, trening oraz test sieci.

Poniżej przedstawiamy wyniki przeprowadzonych przez nas badań skuteczności przewidywania sieci oraz porównanie wykorzystanych narzędzi.

7.1. Wyniki sieci neuronowej

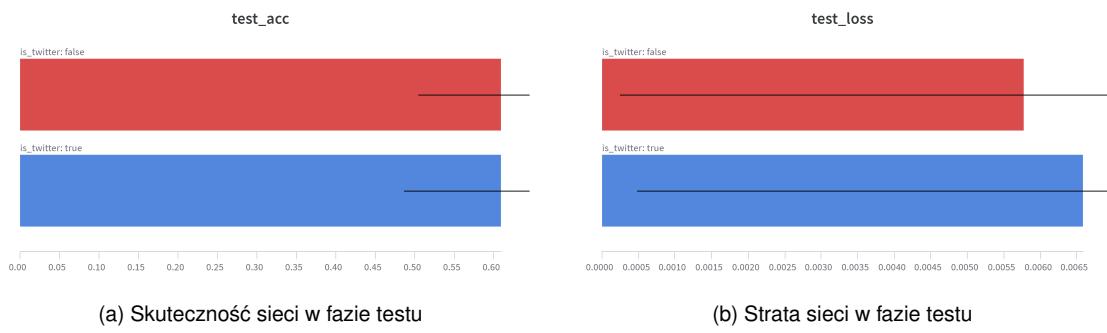
Przedstawiając wyniki naszej sieci porównaliśmy skuteczność modeli w zależności od podanego na wejściu aktywa i sentymentu. Skuteczność rozumiemy zgodnie z opisem z algorytmu 2. W celu uwiarygodnienia wyników, szkoliliśmy modele przy pomocy optymalnej konfiguracji z tabeli 4.1. Przeprowadziliśmy treningi dla wszystkich zgromadzonych przez nas zbiorów danych sentymentu w dwóch wariantach: same dane giełdowe, dane giełdowe wraz z sentymentem. wyniki przedstawiamy na rysunku 7.1



Rys. 7.1. Porównanie skuteczności 12 wybranych przez nas aktywów. aktywa sufixem "-USD" to kryptowaluty

Patrząc na rysunek 7.1 warto zauważać w pierwszej kolejności rozpiętość skuteczności wy-trenowanych modeli. Widzimy wyraźnie, że modele trenowane na danych z rynku kryptowalut mają zdecydowanie gorszą skuteczność w porównaniu do modeli korzystających z danych spółek akcyjnych. Widzimy w ich przypadku także małą korelację między stratą w fazie walidacji, a ostateczną dokładnością. Może to wynikać z natury zachowania kryptowalut, które cechują się nagłymi spadkami i wzrostami. Zastanawiająca może być także różnica w skuteczności modeli w fazie walidacji oraz testu. Różnica ta jest związana najprawdopodobniej z większym podo-

bieństwem podzbioru walidacyjnego niż testowego z podziobrem treningowym. Na wykresie 7.1 możemy zobaczyć także małą różnicę w skuteczności między użyciem danych z *twittera* jak i z giełdy, a użyciem samych danych giełdowych. W celu dokładnego porównania dwóch powyższych metod uśredniliśmy wyniki modeli trenowanych z pomocą danych o sentymencie i trenowanych tylko danymi z giełdy. Wyniki przedstawiliśmy na wykresie 7.2. Na podstawie wykresów przedsta-



Rys. 7.2. Porównanie skuteczności dla uśrednionych danych z sentymensem oraz bez. czarna linia oznacza odchylenie standardowe danego zbioru

wionych na wykresie 7.2 możemy dojść do wniosku że w naszym podejściu dodanie informacji o sentymencie nie wpływa znacząco na wyniki sieci. Średnia skuteczność obydwu podejść plasują się na poziomie 61%, jednak odchylenie standardowe wyników z sentymensem jest nieco większe. Możemy także wywnioskować że modele trenowane bez danych sentymentu mają mniejszą wartość straty w fazie testowej, co bezpośrednio przekłada się na ich większą dokładność.

7.2. Korelacja między sentymensem, a ceną aktywa finansowego

W następującej części skupiamy się na porównaniu działania bibliotek, które liczą sentymensem oraz, co ważniejsze, na znalezieniu korelacji między ceną aktywa, a sentymensem.

7.2.1. Testy działania bibliotek, które liczą sentymensem

Gdy na samym początku procesu szukania korelacji testowaliśmy zachowanie biblioteki *VADER* i *Flair* dla kilku popularnych komunikatów, okazywało się, że wyliczony sentyment rzadko jest poprawny. Poniżej porównanie działania bibliotek dla małego, ręcznie stworzonego przez nas zbioru danych:

index	Sentence	Expected result	VADER	Flair
1	"VADER is smart, handsome, and funny!"	positive	0.844	POS: 1.0
2	"VADER is not smart, handsome, nor funny."	negative	-0.742	NEG: 1.0
3	The book was good.	positive	0.44	POS: 0.567
4	At least it isn't a horrible book.	neutral	0.431	POS: 0.807
5	"The plot was good, but the characters are un compelling and the dialog is not great."	neutral (mixed)	-0.704	NEG: 0.998
6	Not bad at all	neutral	0.431	NEG: 0.759
7	"\$AAPL Attempting a breakout to the upside, now green on the day!"	positive	0.0	NEG: 0.99
8	"Apple's, \$AAPL, most in-demand iPhones will fall short of earlier shipment estimates by 6 million units"	negative	0.0	NEG: 0.953
9	"\$AAPL Threatening a red to green move this morning, keep an eye on this descending triangle setup"	neutral	-0.527	NEG: 1.0
10	"Most Traded Contracts \$AAPL November \$150 Call, \$TSLA November \$200 Call, \$AMZN January 2023 \$140 Put"	neutral	0.0	NEG: 0.913

Tabela 7.1. Przykład obliczania sentymentu przez biblioteki *VADER* i *Flair*
źródło: opracowanie własne

Podejrzewamy, że biblioteki te zwracają niepoprawne wyniki, ponieważ zbiory danych czy leksykony, które wykorzystują do obliczeń mocno różnią się od zbioru danych z platformy *Twitter* i należą do specyficznego kontekstu językowego. Model *Flair* jest trenowany na zbiorze danych

IMDB. VADER korzysta z leksykonu i zdefiniowanych zasad. Zbiór danych z *Twittera* osadzony jest w kontekście typowo giełdowym. Dobrym pomysłem byłoby wytrenowanie modelu, z którego korzysta *Flair* na zbierze danych z *Twittera*, który stworzyliśmy. Jednak w takim podejściu dużym problemem jest brak sklasyfikowanego zbioru treningowego. Z kolei chcąc poprawić wyniki *VADER* musielibyśmy stworzyć dedykowany leksykon.

7.2.2. Korelacja między sentymentem , a ceną danego aktywa

Sprawdziliśmy, czy istnieje korelacja między sentymentem wyliczonym z tweetów, a ceną danego aktywa. Dla pobranych tweetów zawierających nazwę firmy i jej symbol przeprowadziliśmy następujący eksperyment:

1. wyliczyliśmy sentyment każdego tweetu przy pomocy bibliotek *Vader* i *Flair*
2. pobraliśmy dane giełdowe wybranego aktywa
3. połączyliśmy je ze sobą w strukturę danych *Dataframe* z biblioteki *Pandas*
4. stworzyliśmy wykres pokazujący zależność między sentymentem, a ceną aktywa

Proces ten był niezwykle czasochłonny, ponieważ uruchomiony na standardowym komputerze dla danych pojedynczej firmy z danymi z okresu ośmiu lat trwał trzy godziny. Było to spowodowane powolnym działaniem biblioteki *Flair*. Po każdym uruchomieniu tworzyliśmy plik CSV by w razie zmiany sposobu rysowania wykresu nie powtarzać długiego etapu wyliczania sentymentu. Skutecznym rozwiązaniem tego problemu mogłoby być zrównoleglenie przetwarzanie.

W celu porównania wartości sentymentu wyliczonych dla tweetów przez obydwie biblioteki musielibyśmy ujednolicić ich ocenę. *VADER* zwraca wynik w postaci pokazanego poniżej słownika:

```
{'pos': 0.746, 'compound': 0.8316, 'neu': 0.254, 'neg': 0.0}
```

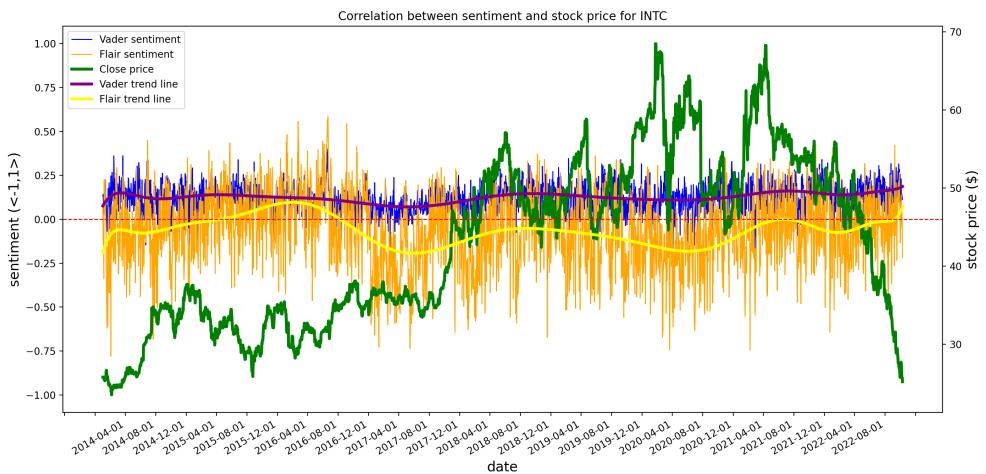
Gdzie *compound* oznacza sentyment wiadomości wyliczony na podstawie wartości *pos*, *neu*, *neg* i mieści się w zakresie $< -1, 1 >$. *Flair* zwraca wynik w postaci pokazanej poniżej klasy z polami:

```
value: POSITIVE
score: (0.9961)
```

W celu łatwego porównywania postanowiliśmy przeskalać rezultat zwrócony przez klasyfikator *Flair*. Jeśli zwrócony wynik jest negatywny to wartość przemnażamy przez -1 . Ostatecznie wynik mieści się zakresie $< -1, 1 >$ i może być porównywany z tym zwróconym przez *VADER*.

Cena giełdowa danego aktywa nie jest normalizowana do przedziału $< -1, 1 >$.

Jak widać na poniższym wykresie dane nie są skorelowane. Sentyment wyliczony przy użyciu biblioteki *Flair* ma tendencję do bardzo skrajnego klasyfikowania, przez co jego linia wydaje się nieustannie skakać. Możliwe, że klasyfikator wychwytuje kluczowe słowa i na tej podstawie przydziela skrajną wartość. Z kolei linia sentymentu biblioteki *Vader* jest skupiona wokół wartości neutralnych i niskich pozytywnych. Może to wynikać z faktu, że sentyment długich wiadomości jest uśredniany.



Rys. 7.3. Wykres korelacji między sentymentem, a ceną aktywa

W poniższej części prezentujemy nasze obserwacje i możliwe dalsze propozycje rozwoju.

Ostatecznie model naszego rozwiązania składa się z jednego repozytorium kodu, w którym znajdują się trzy podprojekty:

- aplikacja użytkownika (*frontend*)
- projekt z logiką modelu sieci neuronowej oraz obsługujący zapytania aplikacji użytkownika o dane takie jak np. wcześniej wytrenowane modele (*backend*)
- projekt z logiką pobierania danych z *Twittera* oraz przetwarzaniem ich

7.3. Możliwe usprawnienia

W poniższych punktach przedstawiamy możliwe ścieżki dalszego rozwoju naszego projektu.

7.3.1. Możliwe usprawnienia w zakresie sieci neuronowych

Dobrym pomysłem na sprawdzenie jakości działania wybranej sieci i wyeliminowanie potencjalnych błędów jest porównanie kilku różnych typów sieci neuronowych. Na dobry początek wartoymi uwagi wydają się być:

- sieć neuronowa *RNN*
- samoorganizująca się rozmyta sieć neuronowa (*SO-NFS*)

Pierwsza z sieci jest pokrewna z siecią *LSTM*. Jest również rekurencyjna. Użycie drugiej pozwala osiągać skuteczność w przewidywaniu spadku bądź wzrostu aktywa na poziomie 87% (źródło).

7.3.2. Możliwe usprawnienia w zakresie użytych danych

Wykorzystanie indeksu **DIJA** lub **S&P 500**

DIJA wskaźnik kursów akcji 30 największych amerykańskich korporacji przemysłowych. **S&P 500** to indeks giełdowy, w skład którego wchodzi 500 przedsiębiorstw o największej kapitalizacji, notowanych na giełdach *New York Stock Exchange* i *NASDAQ*. Wskaźnik ten jest powszechnie uważany za jeden z najlepszych wskaźników dużych amerykańskich akcji (źródło). Przedstawione wskaźniki odzwierciedlają zmiany i trendy gospodarcze, więc ich wykorzystanie w połączeniu z danymi giełдовymi mogłyby poprawić skuteczność przewidywania.

Zautomatyzowanie procesu tworzenia bazy danych z tweetami

Uruchamianie w chmurze procesu z pobieraniem tweetów pozwoliłoby na wygodne jednorażowe uruchomienie procesu. Jednak zmniejszyłoby to kontrolę nad pobieranymi danymi. W czasie pobierania zdarzały się sytuacje, w których wyszukiwanie przypadkowo dotyczyło dużo szerszego kontekstu niż powinno, przykładowo symbol giełdowy firmy *Spotify* to *SPOT*, a po angielskie słowo *spot* oznacza miejsce lub plac. W efekcie pobieranych tweetów było bardzo dużo i co gorsza tematycznie nie dotyczyły one żądań. Nie bez znaczenia pozostaje również miesięczny limit możliwych do pobrania tweetów, czyli 10 milionów.

Takie podejście z pewnością oszczędziłoby nam sporo czasu, ale też wymagałoby wnikliwego przeanalizowania fraz będących częścią filtrowania. Poza tym pozostaje kwestia uruchomienia aplikacji w chmurze, a to potencjalnie wiąże się z kosztami.

Podejście do danych giełdowych

Bezsprzecznie uzupełnianie danych giełdowych o wartości brakujące prostą interpolacją sprzyja pracy nad projektem. Nie trzeba się zastanawiać co zrobić z brakującymi danymi, albo w drugą stronę - jak wyciąć dane z *Twittera*, które nie są potrzebne, bo przykładowo, giełda amerykańska nie działa w *weekend*. Jednak tutaj pojawia się pytanie, czy większe znaczenie w kontekście przewidywania mają dane giełdowe, czy sentyment. Jest to z pewnością interesujące pole do przyszłych eksperymentów.

7.3.3. Możliwe usprawnienia w zakresie analizy sentymentu

Istnieje wiele podejść do przygotowania danych sentymentu. Oprócz wykorzystania algorytmu, bądź sieci neuronowej do ich przetwarzania, można stosować wstępne oczyszczenie danych. W następnych sekcjach przedstawiamy możliwe inne ciekawe podejście do tego zagadnienia, których nie wprowadziliśmy.

Wykorzystanie wszystkich tweetów, nie tylko zawierających daną nazwę firmy

Ciekawym podejściem wydaje się zbadanie społecznego sentymentu całościowo, nie tylko dla wybranej firmy. Podejście takie wymaga cierpliwości w pobieraniu tweetów (przypomnijmy: w ciągu sekundy dodawanych jest średnio sześć tysięcy tweetów). W ciągu dnia publikowanych jest $24 * 60 * 60 * 6000 = 518,4M$ tweetów. Pobranie 500 tysięcy tweetów zajmuje około godziny. Więc stworzenie sensownej bazy danych z tweetami z, przykładowo, kwartału ($518,4M * 90 = 45,66G$), zajęłoby 93312 godzin, czyli 3888 dni.

Oczyszczenie danych z Twittera

Jednym z najprostszych i zarazem skuteczniejszych sposobów na poprawę działania może być dodanie specjalnego, wstępnie przygotowania tweetów, które zostały wykorzystane w pracy *Predicting Stock Movement Using Sentiment Analysis of Twitter Feed with Neural Networks*. Twórcy artykułu poddają każdy tweet specjalnemu procesowi, który sprawia, że są lepiej analizowane pod kątem sentymentu. Proces ten składa się z następujących kroków:

- każdy tweet jest konwertowany na małe litery
- linki zaczynające się od "http" lub "https" lub "www" są zastępowane przez frazę "URL"

- zastępowanie każdego *emoji* poprzez użycie uprzednio zdefiniowanego słownika zawierającego *emojis* wraz z ich znaczeniem. Przykładowo *emoji ":"* jest zamieniane na słowo "*EMOJIsmile*")
- zastępowanie nazw użytkowników @Usernames słowem "*USER*"
- zastępowanie znaków spacement z wyjątkiem cyfr i liter alfabetu
- trzy lub więcej kolejnych liter jest zastępowanych przez dwie litery. Przykładowo "Haloooo" na "Haloo"
- eliminowanie słów mających mniej niż dwa znaki
- eliminowanie tzw. *stopwords*. *stopwords* to angielskie słowa, które nie dodają wiele znaczenia do zdania. Można je bezpiecznie zignorować bez poświęcania znaczenia zdania. Przykładowo *the, he, have*
- przeprowadzenie lematyzacji. Lematyzacja to proces przekształcania słowa do jego formy podstawowej. Przykładowo ze słowa "lepszy" na słowo "dobry"

Dodanie zautomatyzowanego filtrowania kontekstowego tweetów

Istnieje plik .csv zawierający wyodrębnione kategorie, po których można filtrować tweety. (<https://raw.githubusercontent.com/twitterdev/twitter-context-annotations/main/files/evergreen-context-entities-20220601.csv>). Fragment pliku:

domains	entity_id	entity_name
"131,166"	1303008576286281742	\$AMZN
"47,131"	10051086127	Etsy
"131,166"	1301195966125494272	\$BTC
131	864154902926196737	S&P 500

Skorzystanie z takiego filtrowania, zapewnia wyższe dopasowanie pobieranych tweetów. Problemem jest to, że nie zawsze istnieje kontekst dla dowolnie wybranego aktywa finansowego.

Analizowanie komentarzy do tweetów

Analizując komentarze do tweetów uzyskalibyśmy sporo dodatkowych danych, ponieważ tweety uzyskują około pięciu odpowiedzi (źródło). W łatwy sposób można je pobrać korzystając z pola *conversation_id*. Podsumowując, gdybyśmy dla każdego tweetu pobrali również jego odpowiedzi, to mielibyśmy pięć razy więcej danych do przetworzenia. To z kolei mogłoby przełożyć się na bardziej precyzyjne wyniki przewidywania.

7.4. Wnioski na temat wyboru architektury

Realizując projekt zdecydowaliśmy się na początku na architekturę mikroserwisów, gdzie każdy mikroserwis zawiera się w osobnym repozytorium kodu. Przestrzeń nazw i plików nie jest współdzielona, a każda nowa, a zarazem odrębna funkcjonalność realizowana jest jako nowy mikroserwis. W zamian za to występuje duża separacja logiki (co podnosi czytelność) oraz duża rozszerzalność kodu.

Początkowo takie podejście pozwalało skupić się na rozwijaniu poszczególnych serwisów, a raz stworzony interfejs komunikacyjny (protokół *HTTP*) rzadko ulegał zmianie. Jednak wraz z dodawaniem nowych funkcji do naszego projektu i próbami interakcji między istniejącymi komponentami, okazało się, że dalsze rozwijanie projektu jest niewygodne. Uruchamianie kilku aplikacji, było czasochłonne i wymagające dla przeciętnego komputera typu PC. Kluczową decyzją było

oszacowanie przyszłej wielkości projektu i połączenie pokrewnych mikroserwisów w jedno duże repozytorium. Przyniosło to wymierne korzyści:

- Usprawniło proces uruchamiania. Nie musielibyśmy uruchamiać kilku *IDE*
- Pomogło zlokalizować niepotrzebne części projektu (nie potrzebowaliśmy mikroserwisu do pobierania danych giełdowych - wystarczył moduł *Yahoo Finance*)
- Uprościło przesyłanie zmian do zdalnego repozytorium

Zauważylismy, że w zespole dwuosobowym, przy dobrym podziale obowiązków niezwykle rzadko rozwija się jednocześnie te same fragmenty kodu, więc rzadko występują konflikty. Takie podejście do projektu usprawnia pracę.

Wykorzystując narzędzie kontroli wersji *Git* można w łatwy sposób połączyć osobne repozytoria zachowując ich historię. Następująca seria komend pozwala włączyć *projekt A* do *projektu B*:

```
1 cd path/to/project-b
2 git remote add project-a /path/to/project-a
3 git fetch project-a --tags
4 git merge --allow-unrelated-histories project-a/master
5 git remote remove project-a
```

Listing 7.1: komendy pozwalające na połączenie dwóch repozytoriów *Git*, źródło: StackOverflow

WYKAZ LITERATURY

- [1] Jaeheon Chun, Jaejoon Ahn, Youngmin Kim, and Sukjun Lee. Using deep learning to develop a stock price prediction model based on individual investor emotions. 2020.
- [2] Nan Jing, Zhao Wu, and Hefei Wang. A hybrid model integrating deep learning with investor sentiment analysis for stock price prediction. 2021.
- [3] Sai Vikram Kolasani and Rid Assaf. Predicting stock movement using sentiment analysis of twitter feed with neural networks. 2020.
- [4] Ramon Lawrence. Using neural networks to forecast stock market prices. 1997.
- [5] Anshul Mittal and Arpit Goel. Stock prediction using twitter sentiment analysis. Curran Associates, Inc., 2020.
- [6] Tim Salimans and Durk P Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016.
- [7] Bing Yang, Hao Jiankun, and Zhang Sichang. Stock market prediction using artificial neural networks. 2012.

DODATEK A: LISTINGI

```
1 FROM python:3.10.0
2 ENV PYTHONDONTWRITEBYTECODE=1
3 ENV PYTHONUNBUFFERED=1
4
5 COPY requirements.txt requirements.txt
6 RUN --mount=type=cache,target=/root/.cache/pip pip install -r requirements.txt
7 COPY .
8 ENV FLASK_APP=main.py
9 CMD python -u src/main.py
```

Listing 7.2: zawartość pliku *Dockerfile*

```
1 version: '3.7'
2
3 services:
4   frontend:
5     container_name: frontend
6     image: frontend:latest
7     networks:
8       - sentimental-investors-net
9     ports:
10      - "3000:3000"
11     build:
12       context: ../Frontend
13       dockerfile: Dockerfile
14     environment:
15       - REACT_APP_DOCKERENV=true
16       - NODE_OPTIONS=--openssl-legacy-provider
17
18   rnn:
19     container_name: rnn
20     image: rnn:latest
21     networks:
22       - sentimental-investors-net
23     ports:
24      - "4000:4000"
25     build:
26       context: ../RNN
27       dockerfile: Dockerfile
28     volumes:
29       - sentimental-data:/dataset:rw
30       - sentimental-models:/models:rw
31     environment:
32       - WANDB_API_KEY=
33       - RNN_APP_DOCKERENV=true
34
35   engine:
36     container_name: engine
37     image: engine:latest
38     networks:
39       - sentimental-investors-net
40     ports:
```

```

41     - "5000:5000"
42
43   build:
44     context: ../Sentimental-engine
45     dockerfile: Dockerfile
46     volumes:
47       - sentimental-data:/src/data:rw
48
49 networks:
50   sentimental-investors-net:
51
52 volumes:
53   sentimental-data:
54   sentimental-models:

```

Listing 7.3: zawartość pliku *docker-compose.yaml*

```

1 import React from "react";
2 import "../styles/navBar.css"
3
4 function Navbar() {
5   return (
6     <div className="navigation">
7       <ul className="myUL">
8         <li><a className="active" href="/about">About</a></li>
9         <li><a href="/nnModelForm">Create own model</a></li>
10        <li><a href="/predictAssetPrice">Predict</a></li>
11        <li><a href="/createOwnTwitterDataSet">Create own data set</a></li>
12      </ul>
13    </div>
14  );
15}
16
17 export default Navbar;

```

Listing 7.4: Komponent funkcyjny

```

1 import React, { Component } from "react";
2 import DatePicker from "react-datepicker"
3
4 export default class MyDatePicker extends Component {
5
6   currentDate = new Date();
7
8   constructor(props) {
9     super(props);
10    this.state = {
11      isDateValid: true
12    }
13
14    this.handleDateChange = this.handleDateChange.bind(this);
15  }
16
17  isDateValid = (date) => date !== null || date <= this.currentDate;
18
19  handleDateChange(dateAsEvent) {
20    this.props.onDateChange(dateAsEvent);
21    this.setState({ isDateValid: this.isDateValid(dateAsEvent) });

```

```
22    }
23
24    render() {
25        return (
26            <>
27                <div id="date_picker_div">
28                    <label>{this.props.labelBeforeDate}</label>
29                    <DatePicker
30                        value={this.props.date}
31                        onChange={this.handleDateChange}
32                    />
33                    {(!this.state.isValid) ? <><br /><span style={{ color: "red"
34 " }}>Invalid date!</span></> : <></>}
35                </div>
36            </>
37        )
38    }
}
```

Listing 7.5: Komponent klasowy