STŘEDOŠKOLSKÁ ODBORNÁ ČINNOST

Obor č. 18: Informatika

Využití strojového učení pro predikci vývoje cen akcií na světovém trhu

STŘEDOŠKOLSKÁ ODBORNÁ ČINNOST

Obor č. 18: Informatika

Využití strojového učení pro predikci vývoje cen akcií na světovém trhu

The use of machine learning for prediction of stock price development on the world market

Autor: Miroslav Jarý

Škola: Jiráskovo gymnázium, Náchod, Řezníčkova 451

Kraj: Královehradecký

Konzultant: RNDr. Jan Preclík, Ph.D.

Náchod, 2020

PROHLÁŠENÍ

Prohlašuji, že jsem svou práci SOČ vypracoval samostatně a použil jsem pouze prameny a literaturu uvedené v seznamu bibliografických záznamů.

Prohlašuji, že tištěná verze a elektronická verze soutěžní práce SOČ jsou shodné.

Nemám závažný důvod proti zpřístupňování této práce v souladu se zákonem č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších předpisů.

V Náchodě dne 25.03.2020 _____

(Jarý Miroslav)

PODĚKOVÁNÍ

Rád bych tímto poděkoval především mému konzultantovi, RNDr. Janu Preclíkovi, Ph.D., za veškerou podporu projevenou v průběhu vývoje této práce, a za jeho přístup k popularizaci počítačových věd, která významně přispívá k růstu zájmů o tyto obory v našem regionu. Dále bych rád poděkoval Mgr. Petře Půlpánové za jazykovou korekturu této práce.

ANOTACE

Tato práce se zaměřuje na využití strojového učení v praxi. Cílem bylo vytvořit aplikaci, která dokáže několika různými metodami předpovídat vývoj cen akcií na světovém trhu, a následně tyto metody porovnat a zhodnotit jejich rychlost, efektivitu a/nebo jiné parametry.

KLÍČOVÁ SLOVA

strojové učení, Python, vývoj cen

ANNOTATION

This work is focused on usage of machine learning in practice. The main goal was to create an application, which can use several methods to predict stock prices on the global market, and then compare these methods and evaluate their speed, effectivity and/or other parameters.

KEYWORDS

machine learning, Python, price development

OBSAH

1 ÚVOD

Počítače jsou již od druhé světové války nezbytným pomocníkem při každém lehkém či těžkém úkonu. Jako jeden z příkladů můžeme uvést počítač Colossus, který byl sestrojen roku 1943 spojenci ve Velké Británii. Tento přelomový stroj, mající 1 500 elektronek, což bylo desetkrát více, než měl dosud nejvýkonnější stroj, zkrátil dobu dešifrování zpráv nacistického Německa z týdnů na pár hodin. (1)

Avšak výpočetní techniku nemůžeme vnímat jen jako zbraň; dnes je nedílnou součástí našeho každodenního života. Zajeďte si do nejbližšího města, najděte libovolnou školu a vstupte do třídy – u katedry, kde bývala v pevných deskách třídní kniha, naleznete stolní počítač. Nad tabulí bude jasně bílá plocha a nad hlavami žáků zavěšen projektor. Ani děti se nechovají stejně jako dříve – pokud by zrovna neprobíraly odpolední zápas ve fotbale, budou mlčky sedět v lavicích, a na svých mobilech s nadšením hrát novou hru nebo procházet internet.

Věda je však dvojsečný meč; vytvoří stejné množství problémů, jaké vyřeší. Nezbývá nám tedy nic jiného než se s tímto faktem smířit a snažit se z přinesených výhod vytěžit co nejvíce. A také proto můžeme vidět prolínání výpočetních technologií do nejrůznějších oborů, od simulací fyzikálních jevů, až po účetní systémy v malých i velkých firmách.

A právě jedním z takovýchto prolnutí se zabývá tato práce. V té se rozhodně nepředpokládá, že úspěšně předpovíme chování ceniny či jiné komodity v blízké budoucnosti. Cílem je spíše ukázat, jak nám technologie v takových předpovědích mohou pomoci. Ať už ukázkou, co má na trh opravdový vliv nebo jen nastíněním možného budoucího vývoje.

2 STROJOVÉ UČENÍ A JEHO SOUČASNÁ PODOBA

2.1 Definice a historie

2.1.1 Počátky strojového učení

Termín „strojové učení“ (angl. Machine Learning) definoval v roce 1952 zaměstnanec tehdejšího lídra počítačových věd IBM Arthur Lee Samuel. (2) Ten byl před druhou světovou válkou známý hlavně svým nemalým podílem na vývoji klystronů – elektronek, používaných ke generování a zesilování mikrovln. Avšak kvůli nedostatečnému platovému ohodnocení a následnému znechucení práce se vydává na akademickou půdu, přesněji na univerzitu v Illinois. Tato prestižní škola, nacházející se mezi městy Urbana a Champaign, se již dlouhou dobu potýkala s chybějící výpočetní technikou pro zdejší výzkumy. Samuel se po příchodu rozhoduje sestavit pro univerzitu počítač vlastnoručně, a od vedení na tento projekt získává 110 tisíc dolarů, což by v dnešní době odpovídalo částce převyšující 26 milionů korun.

Obrázek 1: Arthur Lee Samuel hrající šachy s počítačem společnosti IBM

Avšak na sklonku roku 1948 projektu rychle peníze dochází, a tak vedení univerzity navrhuje, aby byl celý projekt popularizován a přitáhl pozornost investorů. Po dlouhém přemítání se Samuel nechává inspirovat prací jednoho z největších informatiků Clauda Shannona, který usiloval o počítač schopný hrát šachy, a tím vzniká odnož projektu, zaměřená na sestavení levného výpočetního stroje schopného hrát dámu. Bohužel celá práce, která měla rozpočet zachránit, však zarazila poslední hřebík do rakve, a celý projekt končí fiaskem. Po něm se Samuel přesouvá do již zmíněné firmy IBM. Zde je zařazen do stavby nového počítače IBM 701, prvního komerčně zaměřeného počítače pro vědecký výzkum. A právě pro tento stroj Samuel začíná od píky vymýšlet vysněný algoritmus pro hraní dámy. (3)

Větších úspěchů však dosáhl v konkurenčních Bellových laboratořích v New Jersey již zmíněný Claude Shannon. Ten kromě své práce „Programming a Computer for Playing Chess“ sestrojil roku 1950 jeden z prvních příkladů využití strojového učení v praxi – bludiště sestávající se z pětadvaceti polí a myš jménem Théseus. Ta se skládala z dřevěného šasi, vodivých vousků a magnetu vespod. S ním bylo pohybováno za pomoci elektromagnetu umístěného pod deskou bludiště a napojeného na několik motorků, které s ním pohybovaly. Podle toho, jak vodivé vousky narážely na ocelové stěny bludiště, tak se počítač, rovněž schovaný pod deskou, učil trasu a snažil se zjistit, jak se co nejsnadněji dostat do cíle. Již po prvním průchodu, který sloužil pro zmapování bludiště, se čas zdolání zmenšil na čtvrtinu. (4)

Obrázek 2: Claude Shannon a interiér jeho bludiště.

2.1.2 Útlum a rozdělení oboru

Konec padesátých let ozvláštnil Frank Rosenblatt, který se rozhodl spojit Samuelovy pokusy o strojové učení s teorií o fungování neuronů v mozku kanadského psychologa Donalda Hebba, a dal vzniknout první neuronové síti, zvané The Perceptron. Její primární účel bylo rozpoznávání obrázků; první pokusy vypadaly velice nadějně. Bohužel síť nezvládla rozpoznat většinu testovaných předloh (včetně obličejů), a to zklamalo mnohé, doteď optimistické nadšence.

Na rozhraní 70. a 80. let se do popředí dostává pojem „umělá inteligence“, do té doby spojován se strojovým učením. Umělá inteligence se totiž doposud vytvářela jen díky předchozímu učení počítače. Na povrch se však dere nový způsob tvorby – pomocí předepsaných logických pravidel a bez jakýchkoliv známek učení. Tato metoda rychle získává na popularitě a zanechává původní postupy daleko za sebou. Mnoho nadšených vědců rychle mění obor a celé odvětví načas upadá do zapomnění. (2)

V tomto období se také do povědomí dostává práce Kunihika Fukushimy, japonského vědce a inovátora zabývajícího se studiem hlubokého učení (Deep Learning), která pojednává o rozpoznávání vizuálních vzorů a obrazců. Tento projekt sklízí mnohem větší úspěch, než její pomyslný předchůdce z padesátých let. Můžeme ho též považovat za praotce všech dnešních sítí rozpoznávajících obrázky, tzv. konvolučních neuronových sítí. (5)

Všemu však v 90. letech vdechuje život nová teorie, tzv. „Boosting“. Tyto algoritmy mají za cíl porovnávat jednotlivé sítě, horší vzorky odstraňovat a lepší dále rozvíjet. Tímto způsobem se celá síť snaží „ponaučit z vlastních chyb“.

2.1.3 Jednadvacáté století

Úsvit nového století přináší využití v nových oborech. Prvním z nich bylo rozpoznávání řeči. V rámci tohoto problému byl vyvinut tzv. LSTM model (Long Short-Term Memory). Ten si dokáže zapamatovat až tisíce předcházejících kroků, což je pro řeč důležitá vlastnost.

Další obor, kde strojové učení zažívá boom, je rozpoznávání fotek a obličejů. K němu přispěla převážně iniciativa amerického vládního programu FRGC (Face Recognition Grand Challenge). Díky této vládní soutěži byly v roce 2006 výsledky rozpoznávání desetkrát lepší než v roce 2002, a dokonce až stokrát lepší než v roce 1995. Některé sítě v testech dokonce předčily lidskou mysl – např. od sebe zvládly rozlišit identická dvojčata; to se nepovedlo ani jednomu z testovaných lidských subjektů. (2)

2.1.4 Dnešní stav

Možná si toho ani nevšímáme, ale část strojového učení je všude kolem nás; ať už jde o asistenta v mobilu, cloudové úložiště nebo fotoaparát. Nejčastěji se s ním setkáme v následujících odvětvích:

- ekonomie (analýza prodejů, úprava cen v závislosti na nabídce a poptávce, ...)
- personalizace (reklamy na internetu)
- komunikace (rozpoznávání hlasu či obrazů)

2.2 Metody strojového učení

2.2.1 Učení bez učitele

První metodou je tzv. „učení bez učitele“ (Unsupervised Machine Learning). To se používá převážně v situacích, kdy nemáme vzorek, jak by výsledek měl vypadat. Můžeme si to uvést na příkladu, kdy jdeme k novému příteli domů na oběd. U nás doma, nebo u našich přátel dobře víme, kde najdeme spižírnu nebo televizi. U nového přítele jsme ale ještě nebyli a zprvu jsme zmateni. Víme, co se v každém bytě nalézá (kuchyň, ložnice, koupelna, ...) a víme, že např. toaletu nenalezneme v kuchyni. Avšak po několika návštěvách postupně zjistíme, ve které skříni nalezneme skleničky. (6)

Tato metoda má několik výhod; v první řadě je mnohem snazší hledat v datech nějaký vzorec (např. že většinou se z chodby dostaneme do koupelny). Dále celá analýza probíhá v reálném čase, a to je u počítačů velice žádané. Na druhou stranu, u této metody nemusí být data dostatečně přesná, jednak kvůli absenci vzorků, jednak kvůli ztrátě výpočetního výkonu na samotné vytváření skupin.

Nejpoužívanějším typem této metody je shlukování (Clustering). Jejím cílem je vzít data a rozdělit je do několika skupin (shluků). Ty se buď můžou navzájem překrývat, nebo mohou být striktně odděleny.

2.2.2 Učení s učitelem

Oproti tomu „učení s učitelem“ (Supervised Machine Learning) je proces, kdy máme rozřazený vzorek dat a snažíme se k němu zařadit data nová. Pokud bychom toto měli vysvětlit na příkladu výše, pak tentokrát jsme k první návštěvě u přítele dostali plán bytu, či dokonce nás bytem provedl a ukázal všechna zařízení a zajímavosti.

Jeden z nejdůležitějších typů učení s učitelem je **klasifikace**. Představme si, že jsme byli na cestě po Evropě, a díky tomu máme mnoho fotografií. Ty bychom rádi zařadili do alba v logických uskupeních. A k tomu můžeme využít klasifikační síť. Nejprve jí dáme vzorek dat – v našem případě ukázku jednotlivých skupin, např. fotky krajin a fotky měst. Síť se bude zaměřovat na detaily, které jednotlivé fotografie charakterizují (např. fotky z přírody jsou barevné a nejvíce jim dominuje zelená, naopak fotky velkoměsta jsou povětšinou nevýrazné a vidíme na nich převážně šedé odstíny). Potom, co síť vzorové fotografie nastuduje, můžeme jí dát naše fotky z cest, a ona se bude snažit je podle hledání detailů zařadit do správné kategorie.

Druhý důležitý typ výuky s učitelem je **regrese**. Vraťme se k našemu dobrodružství po starém kontinentu. Než jsme vyrazili, museli jsme si ve směnárně vyměnit měnu pro každou zemi. Chceme přitom ale co nejvíce ušetřit, a tak denně sledujeme zprávy ohledně jednotlivých kurzů. A v tomto nám může ušetřit práci neuronová sít pracující na bázi regrese. Ta zpracovává číselné hodnoty z reálného světa (ceny, úhrny srážek apod.) a následně s nimi pracuje, učí se na nich a nejčastěji se snaží předpovědět budoucí vývoj. Je jisté, že to jde pouze v omezené míře – neuronová síť se nepodívá doma na zprávy nebo nezměří aktuální tlak ovzduší. Má k dispozici pouze historická

data. Samozřejmě, že lze připojit síť k internetu nebo na tlakoměr. To by ale získávala z okolního světa tolik dat, že vyvození logické předpovědi by bylo otázkou náhody a štěstí. (7)

2.2.3 Zpětnovazební učení

V poslední době se díky stále častějšímu uplatnění strojů ve všech oborech objevuje tzv. „zpětnovazební učení“ (Reinforcement Learning). To se snad nejvíce podobá způsobu, jak se živé bytosti učí – omyly a zkušenostmi. Stroj se totiž na základě dat získaných z předešlých operací sám zdokonaluje. Příkladem z říše zvířat může být trénink mazlíčka. Pokud dáme povel „Sedni!“ a naše zvířátko si opravdu sedne, odměníme ho pamlskem. Ovšem pokud zůstane stát nebo se snad rozeběhne do zahrady, nedostane nic, ba ho můžeme i pokárat. Tak si jeho mozek pojí hlasový příkaz k činnosti. Podobně to funguje i u strojů, které zakládají správné či špatné rozhodnutí v závislosti na:

- **hrubé síle** – Vyzkoušej každou možnost a zjisti, která je nejlepší nebo nejefektivnější.
    - Nevýhodou je množství možností, které by zabralo nerozumné množství času.
- **hodnotách** – Snaž se maximalizovat výstupní data.
    - Výstupní data vždy zobrazují dlouhodobý trend.
- **pravidlech** – Najdi v systému pravidla, která ti pomůžou získat co největší odměnu.
    - Právě tento způsob můžeme vidět u zvířat, ale i lidí.
- **simulacích** – Představ si (nasimuluj), co se stane, pokud provedeš...
    - Stejný problém jako u prvního způsobu, časově velmi náročné.

Další rozdělení můžeme provést podle toho, na jaké podněty síť reaguje. Rozlišujeme **pozitivní** a **negativní** zpětnou vazbu. **Pozitivní** typ se snaží o posílení chování, které bylo provedeno správně, a tak navýšit účinnost. Snadno se ale může stát, že síť „přechválíme“, což celý proces učení degraduje. **Negativní** typ se soustředí na opačný problém; snaží se zabránit špatnému chování. Tímto způsobem síť bude zamezovat chybám, ale nebude navyšovat svou účinnost. (8)

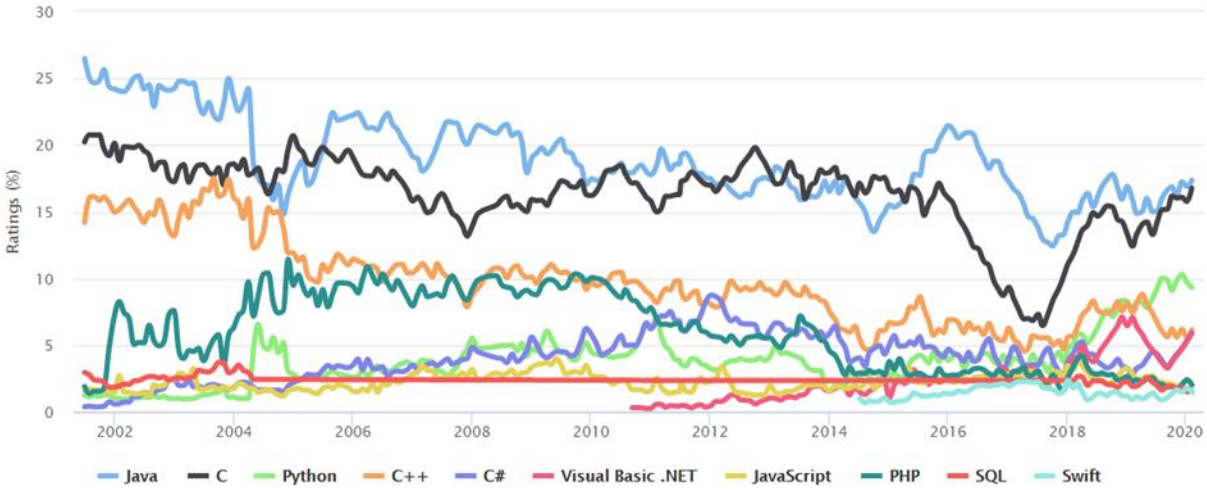
3 VÝVOJ APLIKACE

3.1 Příprava

3.1.1 Výběr prostředí

První důležitou otázkou při přípravě práce byl výběr vhodného programovacího jazyka. Vhodný kandidát by měl mít přehledný kód (kvůli dobré orientaci i pro laika), kvalitní a stabilní API pro strojové učení (balíky knihoven, funkcí, tříd aj., které rozšiřují daný jazyk), a to samé platí i pro samotný jazyk. Do užšího výběru se dostali následující kandidáti:

- **C++** – jeden z nejoblíbenějších programovacích jazyků na světě. Jeho hlavní výhodou je podpora na většině dnešních zařízení a jeho příbuznost s jazykem **C#**, jenž je mi díky dvouletému studiu velice blízký.
- **Python** – často využívaný převážně ve vědeckých kruzích, zato komerční využití je malé. Kód je ale dobře čitelný a orientuje se v něm opravdu každý. Další výhodou je snadná rozšiřitelnost pomocí importovaných knihoven přímo v kódu.
- **Java** – podle indexu TIOBE (9) je Java nejvíce používaný jazyk na světě. Nejčastěji ji můžeme nalézt v podobě skriptů na většině webových stránek. Díky své struktuře a snadnému debuggování má pro tento úkol skvělé předpoklady.



Obrázek 3: Oblíbenost programovacích jazyků v posledních 10 letech

Jako vhodný jazyk byl nakonec vybrán Python z důvodu snadné čitelnosti a rozšiřitelnosti.

3.1.2 Výběr knihoven

Díky výběru Pythonu se vydral na povrch další úkol; vybrat stabilní a dobře vypracované knihovny. Bylo by totiž poměrně hloupé muset programovat některé elementární, ale přesto potřebné funkce. V první řadě muselo být vybráno API pro neuronové sítě. Volba padla na **TensorFlow** a **Keras**.

TensorFlow je platforma pro práci se strojovým učením, jejíž vývoj začal po roce 2010, a to pod záštitou společnosti Google, primárně pro interní účely. Nicméně v roce 2015 vyšla tato knihovna pod licencí Apache 2.0 a v současné době je to bezplatný open-source projekt. (10) Druhý zmíněný modul, **Keras**, je nadstavbou jádra TensorFlow. S ním úzce spolupracuje a uživatel přímo interaguje jen s touto knihovnou, nikoli s jádrem. Modul Keras by se tedy dal nazvat prostředím, ve kterém uživatel jen pokládá jednotlivé vrstvy, určuje počty neuronů atd. (11)

Tyto dvě důležité knihovny dále rozšiřuje balík **sklearn** (celým názvem **scikit-learn**). Jeho historie se datuje do roku 2007, kde vznikl během workshopu *Google Summer of Code* jako projekt Davida Cournapeau. Původně měl pouze rozšiřovat modul **SciPy**, který do Pythonu přináší pokročilou matematiku (např. derivace a integrace), ale i další užitečné nástroje, kupříkladu zpracovávání obrázků. V roce 2010 byl kód zveřejněn a nyní je to osamocený balík funkcí a nástrojů, který se stal jednou z nejpopulárnějších knihoven na platformě GitHub. (12)

Důležité také bylo vybrat nástroj, který bude zajišťovat stahování a aktualizaci dat ze světové burzy. V tomto případě byla vybrána knihovna **yfinance**, která získává data z webu Yahoo Finance. (13) Její současná verze vznikla v roce 2017 jako reakce na ukončení oficiální podpory vlastního API společnosti Yahoo. (14)

Další potřebnou funkcí nezbytnou pro běh kódu jsou tzv. pole a práce s nimi. **Pole** je strukturovaný soubor dat, kde každá hodnota má svou pevně danou pozici, podle které ji můžeme lokalizovat. Jako takové jednoduché dvojrozměrné pole si můžeme představit klasickou tabulku; v ní jsou vypsané různé hodnoty a my přesně určujeme jejich pozici pomocí informace, ve kterém sloupci a řádku se nalézají. Pro tento úkol byl vybrán modul **NumPy** – široce využívaná knihovna v každém kódu, kde se práce s poli vyskytne.

Podobnou funkci, jakou plní **NumPy**, zastává knihovna **pandas**. Ta pracuje s podobnou strukturou dat nazývanou **DataFrame**. Na rozdíl od klasického pole, které se spíše hodí pro tzv. back-end (operace blíže k samotnému procesování), se DataFrame doopravdy blíží tabulce; lze v něm mazat či přidávat sloupce či řádky, v celé tabulce nemusí být jen jeden datový typ (můžeme mít tak první sloupec, který se chová jako text, a zbylé jako čísla, jež se tak i chovají). [15]

Poslední důležitý balík se nazývá **matplotlib** a slouží k vykreslování grafů. Vznikl díky neurobiologovi Johnu D. Hunterovi v roce 2003 a dnes je to nejvyužívanější modul pro vykreslování grafů. Dokonce byl využit při sestavování prvního obrázku černé díry. (16)

3.1.3 Výběr metody neurální sítě

Bod, který zakončuje přípravnou část této práce, je výběr metody, kterou se neurální síť bude učit. Volba padla na učení s učitelem (Supervised Learning) z důvodů vypsaných výše. Důležité je ale také vybrat algoritmus, který síť bude používat ke zpracovávání dat. Na výběr jsou tyto možnosti:

1. **lineární regrese**
    - model se snaží proložit vstupní data přímkou a podle ní určuje výstupní data
    - pro předpověď cen vysoce neefektivní; ceny se nikdy nepohybují lineárně
    - využití převážně u lineárních dějů (např. více peněz = více paliva apod.)

2. **klouzavý průměr**
    - budoucí hodnota se rovná průměru předešlých X hodnot
    - časté využití i mimo strojové učení, převážně na burzách (odhalení trendů)
3. **LSTM (Long Short-Term Memory)**
    - pokročilý typ sítě, schopný zpracovat a zohlednit všechna předcházející data
    - díky své složitosti je zapotřebí velkého výpočetního výkonu
    - použití v rozpoznávání hlasu či obrázků

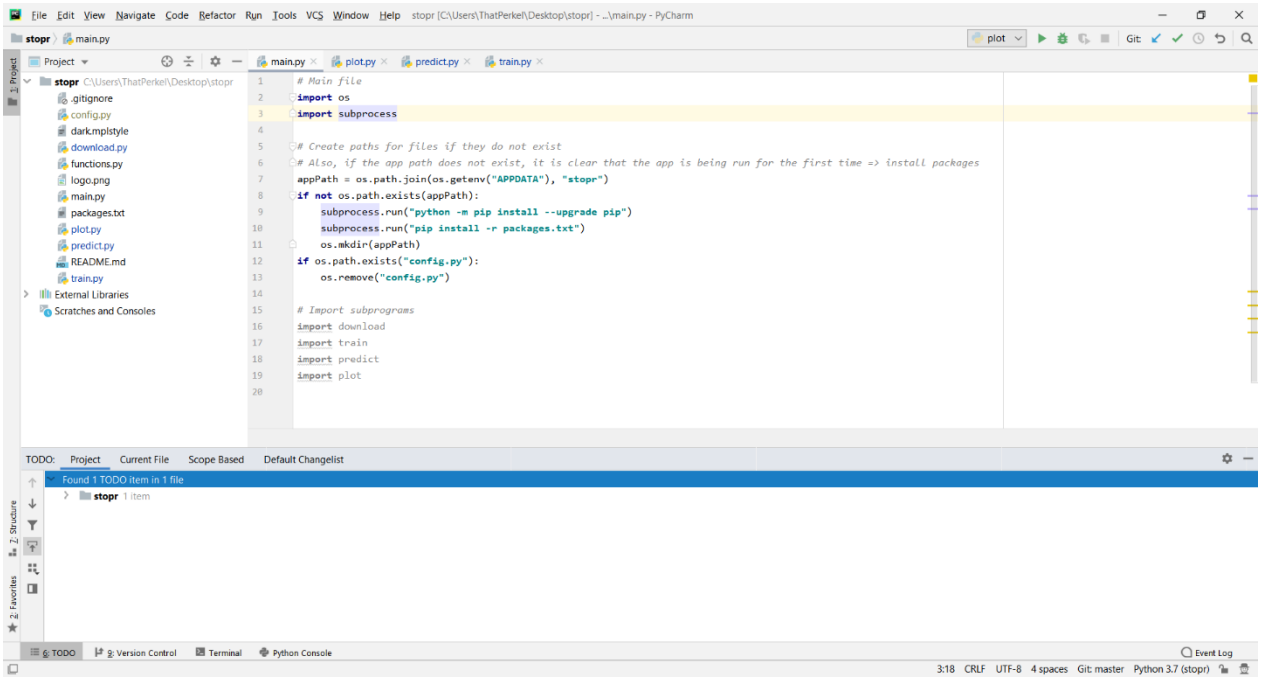
Díky mnohokrát prokázaným excelentním výsledkům **LSTM** sítí jsem vybral právě tuto metodu jako hlavního kandidáta.

3.2 Popis programu

Aplikace, nesoucí pracovní název **stopr** (z angl. **Sto**ck **Pr**ediction), byla z důvodu snazšího vývoje a hledání chyb rozdělena do několika souborů (skriptů).

Obrázek 4: Logo aplikace

Pro vývoj bylo použito prostředí **PyCharm**, které vzniklo v roce 2010 v České republice. Za dobu své existence se stalo velice populární a nyní je vyvíjeno společností JetBrains. Jeho součástí je nástroj pro pokročilé a zároveň snadné debuggování (tj. oprava chyb v programu), virtuální prostředí pro každý projekt, nebo i praktická nápověda při psaní příkazů.



Obrázek 5: Ukázka rozhraní prostředí PyCharm

Koncový uživatel spouští program vždy přes soubor **main.py**, který celou aplikaci řídí. Nejprve kontroluje dle seznamu v **packages.txt**, zdali jsou v systému přítomny všechny potřebné knihovny. Ty stačí nainstalovat pouze při prvním spuštění, proto zjišťuje, jestli existuje složka uživatelských

souborů (jejíž cesta je "*C:\Users**užívatel**\AppData\Roaming\stopr*") a pokud tomu tak není (tzn., že aplikace běží v počítači poprvé), zahájí stahování balíků. Poté již jen kontroluje a spouští ostatní skripty.

3.2.1 Stažení dat

Prvním krokem je samotné stažení dat pomocí knihovny **yfinance** a následná příprava pro neuronovou síť. O to se stará skript **download.py**. Nejprve se uživatele zeptá, na jaké společnosti by rád provedl predikci. Vstupem je standardizovaný index NASDAQ, defaultní hodnotou je **MSFT** (společnost Microsoft).

Pokud je zadaný index platný, skript následně stáhne historická denní data akcií a pomocí knihovny **pandas** je upraví. Nejprve jsou odstraněny dva nepotřebné sloupce – *Adj Close* a *Volume*. První je upravená cena při uzavření trhu za použití různých koeficientů (např. dividend) a druhý ukazuje množství akcií, které je momentálně na trhu. Následně jsou sloupce utříděny v pořadí: *Date*, Open, Low, High, Close. Srovnání lze vidět na obrázku níže. V této tabulce odpovídá sloupec *Date* datu, ze kterého hodnoty jsou, *Open* je cena akcie při otevření trhu (standartně v 9:30 místního času), *Low* je denní minimum ceny, *High* denní maximum ceny, a *Close* hodnota při zavření trhu (standartně v 16:00 místního času). Takto zpracovaná data jsou následně uložena do složky *C:\Users**užívatel**\AppData\Roaming\stopr**index*** do souboru *data.csv*.

Date	Open	High	Low	Close	Adj Close	Volume	Date	Open	Low	High	Close
11/02/2020	190,650	190,700	183,500	184,440	183,938	53 159 900	12/02/2020	185,580	181,850	185,850	184,710
12/02/2020	185,580	185,850	181,850	184,710	184,207	47 062 900	13/02/2020	183,080	182,870	186,230	183,710
13/02/2020	183,080	186,230	182,870	183,710	183,210	35 295 800	14/02/2020	183,250	182,650	185,410	185,350
14/02/2020	183,250	185,410	182,650	185,350	184,845	23 149 500	18/02/2020	185,610	185,500	187,700	187,230
18/02/2020	185,610	187,700	185,500	187,230	186,720	27 792 200	19/02/2020	188,060	186,470	188,180	187,280
19/02/2020	188,060	188,180	186,470	187,280	187,280	29 997 500	20/02/2020	186,950	181,100	187,250	184,420
20/02/2020	186,950	187,250	181,100	184,420	184,420	36 862 400	21/02/2020	183,170	177,250	183,500	178,590
21/02/2020	183,170	183,500	177,250	178,590	178,590	48 572 600	24/02/2020	167,770	163,230	174,550	170,890
24/02/2020	167,770	174,550	163,230	170,890	170,890	68 311 100	25/02/2020	174,200	167,650	174,840	168,070
25/02/2020	174,200	174,840	167,650	168,070	168,070	68 073 300	26/02/2020	169,710	168,210	173,260	170,170
26/02/2020	169,710	173,260	168,210	170,170	170,170	56 206 100	27/02/2020	163,320	157,980	167,030	158,180
27/02/2020	163,320	167,030	157,980	158,180	158,180	93 174 900	28/02/2020	152,410	152,000	163,710	162,010
28/02/2020	152,410	163,710	152,000	162,010	162,010	97 073 600	02/03/2020	165,310	162,310	172,920	172,790
02/03/2020	165,310	172,920	162,310	172,790	172,790	71 030 800	03/03/2020	173,800	162,260	175,000	164,510
03/03/2020	173,800	175,000	162,260	164,510	164,510	71 677 000	04/03/2020	168,490	165,620	170,700	170,550
04/03/2020	168,490	170,700	165,620	170,550	170,550	49 814 400	05/03/2020	166,050	165,690	170,870	166,270
05/03/2020	166,050	170,870	165,690	166,270	166,270	47 817 300	06/03/2020	162,610	156,000	163,110	161,570
06/03/2020	162,610	163,110	156,000	161,570	161,570	72 821 100	09/03/2020	151,000	150,000	157,750	150,620
09/03/2020	151,000	157,750	150,000	150,620	150,620	70 419 300	10/03/2020	158,160	152,580	161,030	160,920
10/03/2020	158,160	161,030	152,580	160,920	160,920	65 354 400	11/03/2020	157,130	151,150	157,700	153,630

Obrázek 6: Srovnání nezpracovaných (vlevo) a zpracovaných (vpravo) dat

Jazyk Python nemá ideálně řešené proměnné použité ve vícero souborech. Tento problém jsem vyřešil pomocí konfiguračního souboru *config.py*, který se nalézá v instalační složce programu a který obsahuje všechny důležité proměnné.

```
# Download, sort and save data if forceUpdate is enabled
print("Update of ticker data in progress . . .")
data = yf.download(ticker, period="max", prepost=True).to_csv()
with open(dataPath, "w+") as f:
    f.write(data)
df = pd.read_csv(dataPath)
df = df[["Date", "Open", "Low", "High", "Close"]]
df.set_index("Date", inplace=True)
df.sort_values(by=["Date"], ascending=True, inplace=True)
df.to_csv(dataPath)

# Write important global variables to configfile
with open("config.py", "w+") as f:
    try:
        config.ticker
    except (NameError, AttributeError):
        f.write("ticker = " + "\"" + ticker + "\"\n")
    try:
        config.devMode
    except (NameError, AttributeError):
        f.write("devMode = " + str(devMode))
writevar("modelType", modelType)
```

Obrázek 7: Ukázka zdrojového kódu (zpracování dat a ukládání proměnných)

Do souboru *config.py* se tedy zapíše použitý index akcie a důležitá proměnná – **modelType**. Ta celé aplikaci sděluje, jaký typ modelu bude používat. Na výběr jsou čtyři možnosti:

1. **Jedna neuronová síť pro celou predikci**
   (síť se naučí předpovědět všechny dny najednou)
2. **Predikce po jednom kroku s využitím předpověděných hodnot**
   (síť předpoví první krok, ten pak využije k předpovězení dalšího kroku)
3. **Predikce po jednom kroku s využitím historických hodnot**
   (síť předpoví první krok, k dalšímu kroku už ale nepoužívá předpovězenou hodnotu, nýbrž historickou)
4. **Jedna neuronová síť pro jednu hodnotu**
   (pro každou hodnotu je natrénovaná samostatná síť)

Podrobný rozbor jednotlivých metod a jejich efektivita se nachází v kapitole 4.

3.2.2 Trénink modelu

Po přípravě dat je již čas na samotnou stavbu sítě. Na tu se zaměřuje skript **train.py**. Nejprve jsou inicializovány tři důležité proměnné a následně jsou zapsány do již zmíněného konfiguračního souboru. Jedná se o:

- **pastSteps** – kolik dní do minulosti má síť zkoumat
- **futureSteps** – kolik dní budeme předpovídat
- **timeShift** – značí časový posun; pokud se například rovná deseti dnům, znamená to, že začátek predikce bude před deseti dny, pokud je nulová, opravdu předpovídáme budoucí data (od dnešního dne)

Pro zápis proměnných je definována funkce **writevar**. Ta se nachází v souboru **functions.py** v adresáři aplikace. Tato funkce má dvě varianty vstupů:

- **writevar(**name**)** – v případě, že chceme dát uživateli možnost vložit vlastní hodnotu
- **writevar(**name, value**)** – v případě uložení výchozí hodnoty definované v programu

Po zápisu proměnných je vzat jeden ze sloupců souboru *data.csv* (v tomto případě *Close*) a jeho hodnoty jsou naškálovány za pomocí knihovny **sklearn** a její funkce **MinMaxScaler**. Neurální síť má totiž mnohem lepší výsledky, pokud pracuje s daty z rozsahu 0 až 1. Samotný skript ale škáluje hodnoty v rozmezí 0,1 – 0,9. Díky tomu si síť zachová efektivitu i pokud budou předpovězené hodnoty vyšší nebo nižší než dosavadní historické maximum či minimum, což je velice důležité.

Naškálované hodnoty jsou následně uloženy jako sloupec *Scaled* zpět do souboru *data.csv* pro pozdější použití. Důležité je ale i uložit samotnou škálu, podle které se data upravila. Výstup sítě po predikci je též škálovaný a tyto hodnoty koncovému uživateli o ceně moc neřeknou. Tato škála se ukládá do stejného adresáře jako data pod názvem *scaler.dat*.

Poté jsou data rozdělena do dvou sad; první sada je určena pro trénink sítě a druhá bude použita přímo pro predikci. Hranici, podle které je rozdělení utvořeno, určuje právě proměnná **pastSteps**. V našem případě se rovná 120 – tedy data mladší než 120 dní budou určena k predikci, zatímco data starší než 120 dní budou použita k trénování a využita v tomto skriptu.

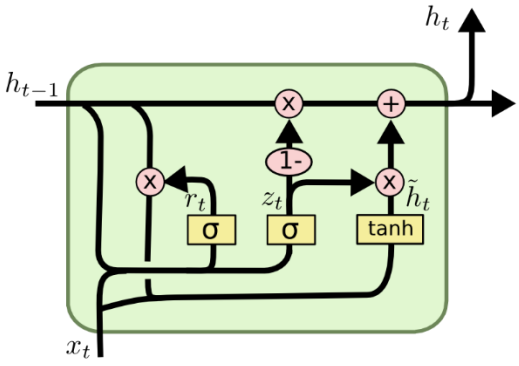
Následuje rozdělení a naformátování dat do dvou polí, jedno pro vstupní hodnoty (historická data) a jedno pro hodnoty výstupní (předpovězená data), které bude síť zpracovávat. Jejich podoba, velikost a rozměry závisí na druhu vybrané sítě a jejich popis lze nalézt v kapitole 4.

V posledním bloku kódu se nachází definice vrstev neuronové sítě. Její složení se liší podle použité metody, běžně zde ale najdeme tři druhy vrstev – LSTM, Dense a Dropout.

```
# Build and compile the model, then save it to TICKER dir
model = Sequential()
if modelType == 1:
    model.add(LSTM(75, return_sequences=True, input_shape=(trainX.shape[1], 1)))
    model.add(LSTM(75))
else:
    model.add(LSTM(futureSteps, input_shape=(trainX.shape[1], 1)))
model.add(Dense(trainY.shape[1]))
model.compile(optimizer="adam", loss="mean_absolute_percentage_error")
for i in range(0, trainX.shape[2]):
    usedX = np.reshape(trainX[:, :, i], (trainX.shape[0], trainX.shape[1], 1))
    usedY = np.reshape(trainY[:, :, i], (trainY.shape[0], trainY.shape[1]))
    if modelType == 4:
        print("\nTraining model for " + str(i+1) + ". step")
    model.fit(usedX, usedY, validation_split=1, epochs=9, batch_size=3)
    if modelType != 4:
        model.save(os.path.join(folderPath, "models", "model.h5"))
    else:
        model.save(os.path.join(folderPath, "models", "model_" + str(i) + ".h5"))
endT = time_ns()
deltaT = str(round(float(endT - startT)/1000000000, 3))
print("Elapsed time: " + deltaT + " s")
```

Obrázek 8: Ukázka kódu (definice struktury neuronové sítě)

LSTM bylo už několikrát v této práci zmíněno, vždy ale pouze okrajově. V zásadě se tato metoda velice blíží způsobu, kterým lidé myslí. Klasické neuronové sítě si nedokážou pamatovat příliš staré informace, v nejhorším případě je zapomenou hned po použití. V tomto směru je LSTM (Long Short-Term Memory) revoluční. Spolu se svými předchůdci patří k tzv. **rekurentním neurálním sítím**. Ty obsahují cykly, které umožňují dlouhodobější uchování informace.



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

*Obrázek 9: Schéma LSTM buňky a zpracování informace (dostupné z https://miro.medium.com/max/1838/1*8nFrwolzTYtUWSaziiJGkg.png)*

Vznik samotných LSTM sítí se datuje do roku 1997 díky práci informatiků Seppa Hochreitera a Jürgena Schmidhubera přímo nazvané „LONG SHORT-TERM MEMORY“. (17) Ta oživila tuto dlouho skomírající oblast informatiky a vývoj neuronových sítí s dlouhodobou pamětí nabral na obrátkách.

Dalším použitým druhem jsou tzv. **Dense vrstvy**. To jsou klasické vrstvy neuronů, které jsou navzájem propojené. Jejich hlavní výhodou je operační rychlost. Nevýhodou může být již zmíněná ztráta dlouhodobé informace, a proto ji používáme zřídka (např. jako výstupní vrstvu).

Posledním druhem jsou **Dropout vrstvy**. Jak již bylo zmíněno v kapitole 2.2.3, zpětnovazební neuronová síť se může „přechválit“ a predikci tak snadno znehodnotit. Tomu se snaží zabránit tato vrstva; část informace, kterou od vrstvy neuronů dostane, vypustí.

3.2.3 Predikce hodnot

Skript předpovídající samotnou předpověď, **predict.py**, se podobá kódu pro trénování. Tentokrát ale bude prvotně naplněno jen jedno pole – pro vstupní hodnoty. Výstupní hodnty bude do druhého pole zapisovat přímo neuronová síť. Podoba polí, a způsob, jakým do něj skript zapisuje, znovu závisí na zvolené metodě a postup je vysvětlen v kapitole 4.

Druhá funkce tohoto souboru je určení střední absolutní procentuální chyby (angl. **MAPE** – Mean Absolute Percentage Error), která je v této práci používána jako měřítko pro porovnání účinnosti jednotlivých sítí. Lze ji vypočítat jako:

$$MAPE(\hat{x}) = 100 \cdot \sum \left(\frac{|x - \hat{x}|}{x} \right);$$

kde $\hat{x}$ je odhadnutá veličina a x reálná veličina. Následným zprůměrováním chyb všech hodnot získáme střední absolutní procentuální chybu celé predikce.

3.2.4 Vykreslení grafu

Po úspěšně dokončené předpovědi je nutné data reprezentovat v grafické podobě. Zde přichází na řadu skript **plot.py** a knihovna **matplotlib**, jež pomocí upraveného stylesheetu (soubor, který udává

vzhled grafu, v tomto případě *dark.mplstyle*, umístěný v instalační složce aplikace) vykreslí historické (modrou barvou) a předpovězené hodnoty (oranžovou barvou).

```
# Getting historic data from "data.csv"
df = pd.read_csv(dataPath)
data = df["Close"].to_numpy()
data = data[len(data)-timeShift-futureSteps:len(data)-timeShift+futureSteps*2]
dataDate = df["Date"].to_numpy()
dataDate = dataDate[len(dataDate)-timeShift-futureSteps:len(dataDate)-timeShift+futureSteps*2]
dataDate = dataDate.astype("datetime64")

# Getting prediction data from "predict.csv"
df = pd.read_csv(predictPath)
predict = df["Value"].to_numpy()
predictDate = df["Date"].to_numpy()
predictDate = predictDate.astype("datetime64")

# Plot the graph with used style
# plt.style.use("dark.mplstyle")
plt.margins(0, 0, tight=True)
plt.rcParams.update({"font.size": 16})
plt.title("Close price of ticker " + ticker)
plt.xlabel("Datestamp", fontsize=14)
plt.ylabel("Close price (USD)", fontsize=14)
historic = plt.plot(dataDate, data, label="Historic data")
predicted = plt.plot(predictDate, predict, label="Predicted data")
plt.tight_layout()
plt.plot()
plt.legend()
mng = plt.get_current_fig_manager()
mng.window.state('zoomed')
plt.show()
```

Obrázek 10: Ukázka kódu (vykreslení grafu)

4 METODY PREDIKCE

Jak již bylo zmíněno v kapitole 3.2.1, v aplikaci jsou zahrnuty čtyři druhy modelů, kde každý z nich musí mít jinak zpracovaná data, vyžaduje jiný výpočetní výkon a i doba samotné predikce se liší. V této kapitole se nachází popis a vysvětlení jednotlivých modelů, principu jejich fungování a jejich vlastnosti, výhody a nevýhody. Výsledky predikcí a jejich srovnání lze nalézt v kapitole 5.

4.1 Okamžitá predikce všech hodnot

Prvním, nejjednodušším způsobem trénování sítě je predikce všech budoucích hodnot naráz. Postup je jednoduchý: síť v tréninkové části dostane dvě pole dat – jedno, ve kterém jsou hodnoty předcházející predikci, a druhé, kde budou hodnoty výsledné. Uveďme si příklad: máme sadu dvaceti hodnot a chceme síť natrénovat, aby z deseti hodnot předpověděla další dvě. Do prvního řádku prvního pole tedy umístíme prvních deset vstupních hodnot a do druhého pole dvě hodnoty výstupní (tj. 11. a 12.). V dalším řádku provedeme tutéž operaci, pouze budeme dosazovat hodnoty posunuté v čase o jednu jednotku. V prvním poli tedy bude 2. až 11. hodnota a v druhém poli 12. a 13 hodnota.

Vstup											Výstup	
1	2	3	4	5	6	7	8	9	10		11	12
2	3	4	5	6	7	8	9	10	11		12	13
3	4	5	6	7	8	9	10	11	12		13	14
4	5	6	7	8	9	10	11	12	13		14	15
5	6	7	8	9	10	11	12	13	14		15	16
6	7	8	9	10	11	12	13	14	15		16	17
7	8	9	10	11	12	13	14	15	16		17	18
8	9	10	11	12	13	14	15	16	17		18	19
9	10	11	12	13	14	15	16	17	18		19	20

Obrázek 11: Ukázka vstupního a výstupního pole pro trénink (okamžitá predikce)

Následně v části pro samotnou predikci dostane síť pouze jeden pomyslný řádek pro predikci (stačí nám totiž předpovědět pouze dvě budoucí hodnoty). To jednoznačně urychluje a zjednodušuje proces samotné predikce; je to taky jeden z důvodů, proč je tato metoda ze všech uvedených nejrychlejší.

4.2 Predikce po jednom kroku

Další metoda může vysvětlit dvě možnosti predikce v aplikaci naráz; jedná se totiž o stejný přístup, liší se pouze použitá data. Jedná se o predikci jen jednoho kroku s použitím historických nebo předpovězených dat. Použijme příklad z předchozí kapitoly; máme sadu dvaceti hodnot a chceme síť naučit z deseti předcházejících hodnot předpovědět dvě následující. Ve fázi učení pouze budeme chtít, aby síť z deseti předcházejících kroků, které umístíme do prvního pole, určila

jedenáctý, patřící do druhého pole. Stejně jako u předcházející metody se s další řádkem posuneme o jeden krok. V druhém řádku tedy bude v prvním poli 2. – 11. hodnota, v druhém poli 12.

Vstup											Výstup
1	2	3	4	5	6	7	8	9	10		11
2	3	4	5	6	7	8	9	10	11		12
3	4	5	6	7	8	9	10	11	12		13
4	5	6	7	8	9	10	11	12	13		14
5	6	7	8	9	10	11	12	13	14		15
6	7	8	9	10	11	12	13	14	15		16
7	8	9	10	11	12	13	14	15	16		17
8	9	10	11	12	13	14	15	16	17		18
9	10	11	12	13	14	15	16	17	18		19
10	11	12	13	14	15	16	17	18	19		20

Obrázek 12: Ukázka vstupního a výstupního pole pro trénink (predikce po jednom kroku)

Více práce však síť čeká při samotné předpovědi. V našem případě totiž proběhne dvakrát (jelikož chceme dvě budoucí hodnoty). Zde lze nalézt jediný, ale podstatný rozdíl mezi využitím předpovězených a historických hodnot.

4.2.1 S využitím předpovězených hodnot

V této metodě se první budoucí hodnota po předpovězení přiřazuje do prvního pole určeného pro historická data. Síť tedy využívá dříve předpovězená data pro další predikci. Nicméně, jak se brzy ukázalo, tento přístup pro predikci není ideální – síť nedokáže do předpovědi zahrnout jakoukoliv změnu ve vývoji (tj. např. změna růstu v pokles) a křivka, kterou data vykreslují, se spíše podobá části hyperboly, než grafu ceny akcie.

4.2.2 S využitím historických hodnot

Mnohem větší úspěch, i když jen zdánlivý, sklízí druhý přístup. V tom již dále nepracujeme s předpovězenými hodnotami a namísto toho k predikci používáme jen data historická. Díky nim může být do sítě zanesena změna ve vývoji a výsledky vypadají mnohem prokazatelněji.

Bohužel, musíme si uvědomit, že tato metoda se dá prohlásit za podvod. Její použití v praxi je totiž nemožné (nemůžeme donekonečna dosazovat historické hodnoty – neznáme je). A právě tento fakt se snaží zamést pod kobereček nespočet webových stránek zabývajících se předpovědí akcií či jiných komodit. (18) (19) Prostému čtenáři, který se nevyzná v kódu, se můžou výsledky zdát zázračné a může rychle usoudit, že se jedná o snadný způsob zbohatnutí. Takovéto stránky zároveň s článkem o „stoprocentně úspěšné předpovědi“ zobrazují odkazy, kde nabízejí své kurzy v programování (samozřejmě za menší finanční obnos).

4.3 Predikce každé hodnoty pomocí samostatného modelu

Poslední metoda by se dala zahrnout do předchozí kapitoly; jedná se totiž také o predikci po jednom kroku, avšak přístup je výrazně odlišný. Zde totiž vytváříme pro každý budoucí krok novou síť, která se na něj soustředí. Znovu to můžeme lépe ukázat na předešlém příkladu: máme dvacet hodnot, a z deseti předešlých předpovídáme dvě budoucí. V tomto případě tedy vzniknou dvě sítě; jedna pro předpověď jednoho kroku dopředu a druhá pro předpověď druhého budoucího kroku.

Vstup 1 & 2											Výstup 1
1	2	3	4	5	6	7	8	9	10		11
2	3	4	5	6	7	8	9	10	11		12
3	4	5	6	7	8	9	10	11	12		13
4	5	6	7	8	9	10	11	12	13		14
5	6	7	8	9	10	11	12	13	14		15
6	7	8	9	10	11	12	13	14	15		16
7	8	9	10	11	12	13	14	15	16		17
8	9	10	11	12	13	14	15	16	17		18
9	10	11	12	13	14	15	16	17	18		19

Vstup 1 & 2											Výstup 2
1	2	3	4	5	6	7	8	9	10		12
2	3	4	5	6	7	8	9	10	11		13
3	4	5	6	7	8	9	10	11	12		14
4	5	6	7	8	9	10	11	12	13		15
5	6	7	8	9	10	11	12	13	14		16
6	7	8	9	10	11	12	13	14	15		17
7	8	9	10	11	12	13	14	15	16		18
8	9	10	11	12	13	14	15	16	17		19
9	10	11	12	13	14	15	16	17	18		20

Obrázek 13: Ukázka vstupního a výstupního pole pro trénink (predikce každé hodnoty pomocí samostatného modelu)

Tento způsob predikce se také velice osvědčil. Každá síť se může soustředit na svoji hodnotu a nemusí se zabývat ostatními. Jak ale bylo zjištěno, některé počítače může zahltit generace tolika sítí (převážně RAM paměť). Proto se tato metoda nedá považovat za efektivní.

5 VÝSLEDKY

5.1 Porovnání rychlosti typů predikce

Každá síť má odlišný přístup k práci s daty a proto se liší i doba trvání provedení předpovědi. Všechny testy byly provedeny pro predikce budoucích deseti hodnot z předchozích dvaceti hodnot. Typy jsou očíslovány podle pořadí v rozhraní aplikace (viz kapitola 3.2.1).

Čas tréninku (v sekundách)

	Test 1	Test 2	Test 3	Test 4	Test 5	**Průměr**
Typ 1	3,428	3,262	3,223	3,296	3,288	**3,299**
Typ 2	3,348	3,346	3,265	3,249	3,300	**3,302**
Typ 3	3,209	3,192	3,309	3,394	3,289	**3,279**
Typ 4	30,070	29,808	29,655	29,315	29,660	**29,702**

Čas predikce (v sekundách)

	Test 1	Test 2	Test 3	Test 4	Test 5	**Průměr**
Typ 1	1,664	1,669	1,667	1,676	1,647	**1,665**
Typ 2	1,783	1,721	1,746	1,732	1,752	**1,747**
Typ 3	1,739	1,765	1,752	1,741	1,725	**1,744**
Typ 4	54,179	53,469	53,399	52,459	52,966	**53,294**

Jak můžeme vidět, časy se u prvních třech typů sítě výrazně neliší. Z toho vyplývá, že čas určený pro trénování a predikci nezávisí na počtu předpovídaných hodnot. U samotné predikce je dokonce čas prvního typu kratší než u následujících dvou typů. To je dáno větším množstvím operací při přípravě dat (viz obrázek níže). U prvního typu sítě se totiž jednou připraví data a jednou předpoví deset budoucích kroků, zatímco u dalších dvou musíme data desetkrát přeorganizovat a desetkrát předpovědět budoucí hodnotu.

Čtvrtý typ se časy vymyká úplně; u tréninku je sice čas zhruba desetkrát větší (což odpovídá předpokladu, trénujeme totiž deset sítí s množstvím dat odpovídajícím typům 2 a 3), ale u predikce je potřebná doba dokonce třicetinásobná. Pravděpodobnou příčinou této extrémní doby je neustálé načítání a mazání jednotlivých modelů, které také zapřičiňuje vysoké využití RAM paměti (v tomto případě se jednalo o 1 GB).

5.2 Porovnání efektivity typů predikce

Důležitým aspektem všech predikcí je jejich přesnost. K tomu byla využita již dříve zmíněná střední absolutní procentuální chyba (viz kapitola 3.2.3). Parametry predikce jsou totožné s předchozí kapitolou.

MAPE epoch při tréninku (v %)

	Epocha 1	Epocha 2	Epocha 3	Epocha 4	Epocha 5	Epocha 6	Epocha 7	Epocha 8
Typ 1	91,2907	46,8823	16,4665	7,901	3,6295	2,597	2,3192	2,0412
Typ 2	38,7618	7,0142	4,6589	1,8379	2,9427	1,4387	1,7227	3,2163
Typ 3	37,6726	9,5612	4,894	2,7352	2,4365	2,1903	3,7479	2,8796
Typ 4 (průměr)	5,5523	2,2987	2,4544	2,6353	2,5607	2,4242	2,2071	2,0227
Typ 4 (posl. síť)	2,2539	2,1189	2,3281	2,2077	3,9094	4,1765	3,9616	3,0321

Můžeme si povšimnout jistého trendu. V první generaci sítí je chyba enormní (40-90%), a v dalších epochách se blíží k jednotkám procent. To je vcelku rozumné, v průběhu učení se síť stále zdokonaluje a zmenšuje své chyby. Tato tabulka však obsahuje dvě anomálie, které jsem pojmenoval **fluktuace chyby** a **paradox následných sítí**.

První problém poukazuje na změny velikosti chyby během evoluce sítě. Patrné to je na datech třetího typu; již druhá generace se dostává k chybě blízké 2%, ale ve dvou následujících epochách se hondota nepatrně zvyšuje. Tato anomálie má jednoduché a logické vysvětlení. Jak již bylo popsáno výše, síť pracuje s velkým množstvím dat. Pokud to zjednodušíme, často se stane, že se síť nepatrně zlepší v předpovědi jedné hodnoty, zatímco se výrazně zhorší predikce hodnoty jiné. Tímto úkonem se tedy celková chyba zvětší.

Paradox následných sítí nastává u čtvrtého typu predikce. Jak již bylo zmíněno, předpovídáme deset budoucích hodnot, u této metody tedy potřebujeme deset různých sítí, každou na jednu hodnotu. Vývoj chyby první sítě vypadá velice podobně jako u typu 2 či 3. Chyba začíná okolo 40%, ale již u čtvrté generace se pohybuje v rozmezí 2-3%. U zbylých devíti sítí tomu tak není – zde se vývoj podobá poslednímu řádku v tabulce, tj. „Typ 4 (poslední síť)“. Sítě žádným způsobem nesdílejí informace o učení, přesto je již chyba první generace blízká 2%. Tento paradox se mi nepodařilo objasnit.

MAPE predikce (v %)

	Test 1	Test 2	Test 3	Test 4	Test 5	**Průměr**
Typ 1	5,48	4,86	5,54	4,8	5,41	**5,22**
Typ 2	6,21	8,29	8,18	4,13	4,57	**6,28**
Typ 3	3,41	3,76	3,66	3,54	3,36	**3,55**
Typ 4	4,78	4,69	3,86	4,24	4,37	**4,39**

U chyby predikcí jsou hodnoty v souladu s předpokladem. Nejnižší chybu (v průměru 3,5%) vykazuje třetí typ sítě, tj. předpověď jednoho kroku za použití historických dat. Z důvodů uvedených v kapitole 4.2.2 však tento typ nelze v praxi využít, a proto ho můžeme z výběru vyřadit.

Druhá nejnižší chyba (v průměru 4,4%) připadá na metodu predikce, kde pro každou hodnotu trénujeme zvláštní síť. Bohužel, z důvodů potřeby vysokého výpočetního výkonu a času pro tuto metodu, ji nelze prohlásit za nejefektivnější. Pokud bychom ale měli neomezený čas pro řešení problému, tento typ predikce by byl preferován.

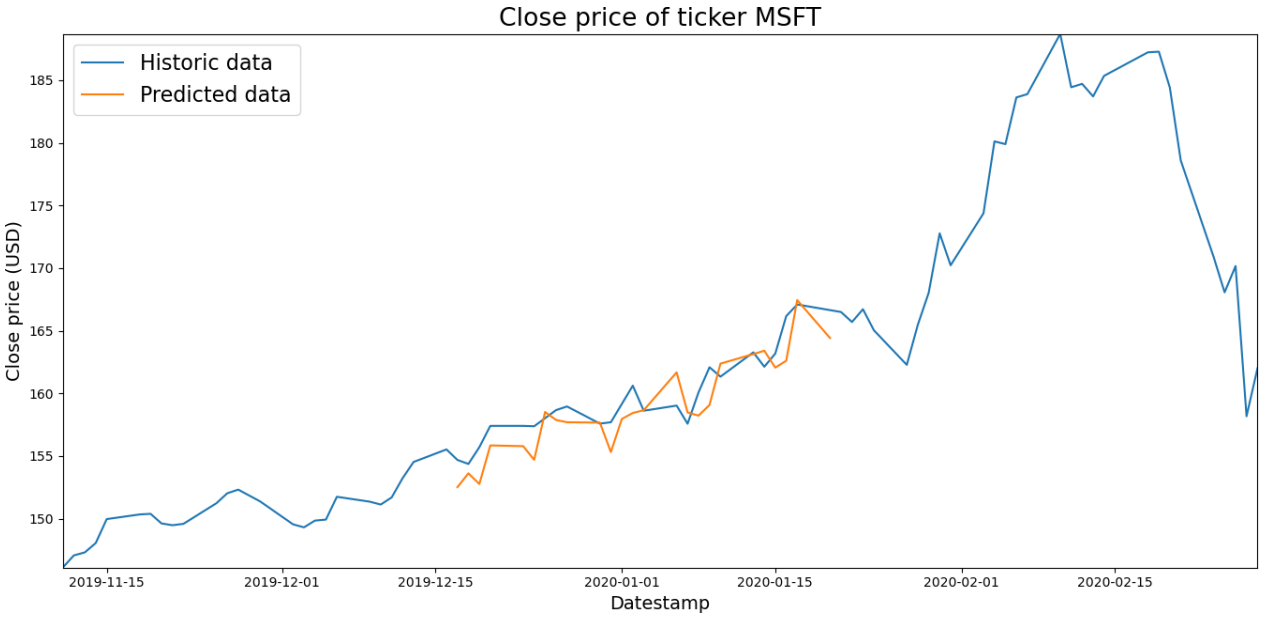
Nejefektivnější metodou (i vzhledem k času) se tedy stává predikce všech hodnot naráz za použití jedné sítě. S chybou 5,2% by se rozhodně dala považovat za užitečnou, i když se nemůžeme na hodnoty stoprocentně spolehnout.

Druhý typ předpovědi (tj. predikce po jednom kroku s použitím předpovězených hodnot) může být prohlášena za nepoužitelnou, závěr se tedy shoduje s informacemi v kapitole 4.2.1.

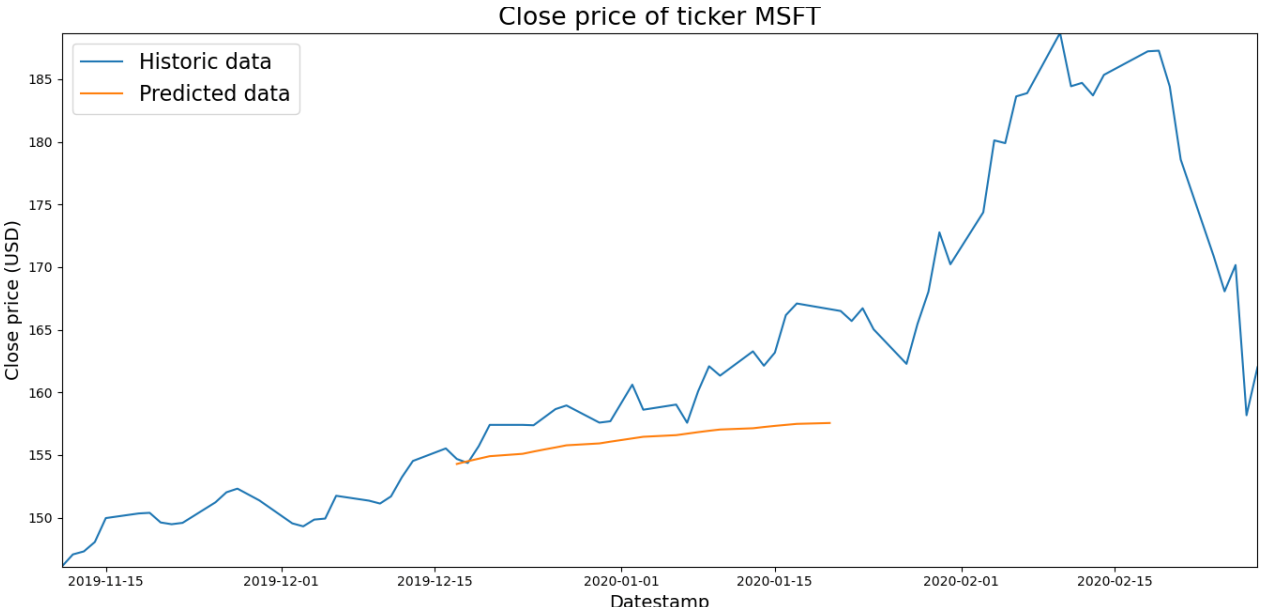
Je také důležité zdůraznit, že ač byly všechny testy provedeny pětkrát, a to na stejném počítači, hodnoty se dají považovat pouze za orientační, a nelze jim přikládat velkou váhu. Jejich pořadí ale můžeme považovat za průkazné.

5.3 Srovnání predikcí

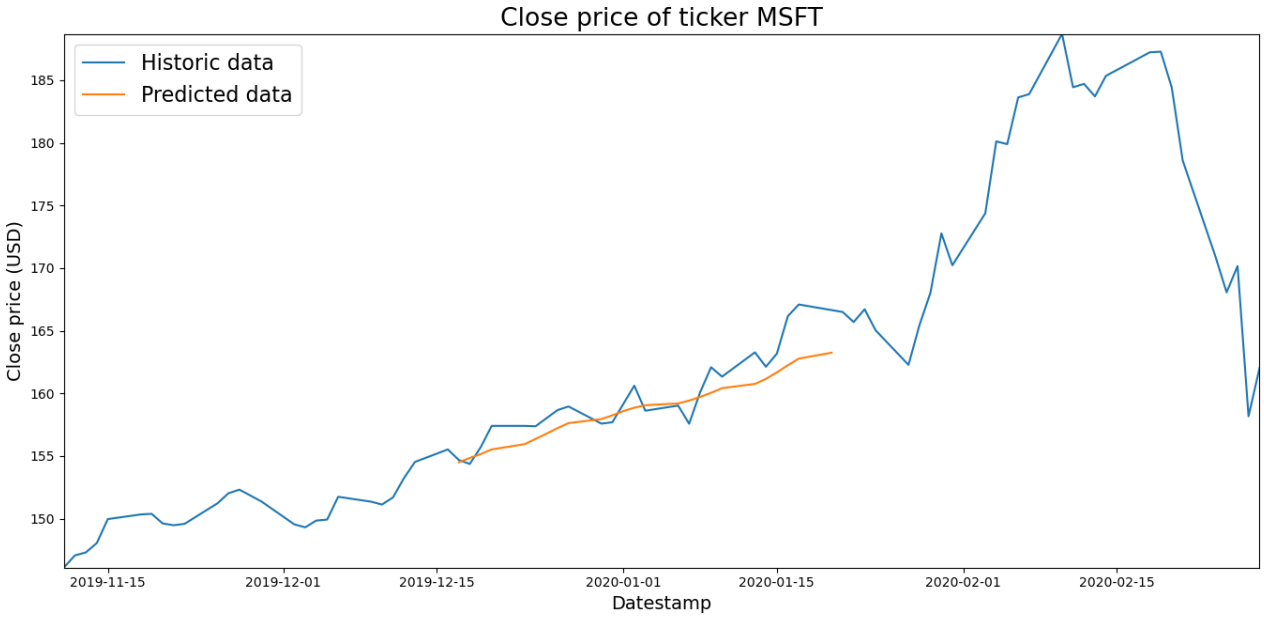
Následující kapitola ukazuje srovnání typů sítí u predikce pětadvaceti budoucích hodnot za pomoci pětadvaceti předchozích hodnot.



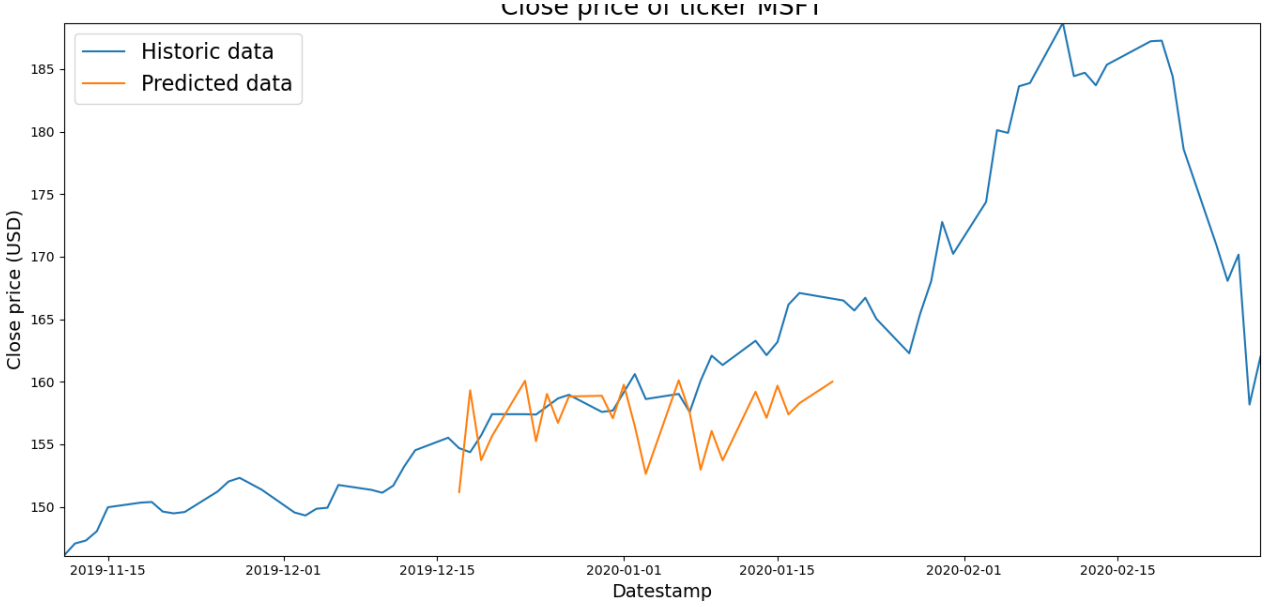
Obrázek 14: Předpověď sítě typu 1 (okamžitá predikce všech hodnot)



Obrázek 15: Předpověď sítě typu 2 (predikce po jednom kroku s využitím předpovězených hodnot)



Obrázek 16: Předpověď sítě typu 3 (predikce po jednom kroku s využitím historických hodnot)



Obrázek 17: Předpověď sítě typu 4 (predikce každého kroku pomocí samostatného modelu)

U typů 2 a 3 vidíme velmi vyhlazené křivky – jak již bylo zmíněno, jelikož je síť naučena předpovědět jen jeden krok, má malý potenciál měnit rychlost růstu či pádu ceny. Stejné je to s lidskými kroky – je velice obtížné udělat půlkrok rychle a půlkrok pomalu. U čtvrtého typu tomu tak není – každý krok je předpovězen jinou sítí, a proto zde můžeme prudké změny pozorovat. První typ také předpovídá výrazné pády a růsty cen – předpovídá totiž celou datovou řadu, stejně jako by mohl lidský mozek naplánovat cestu po chodníku, a každému kroku určil rychlost.

6 ZÁVĚR

Byla vytvořena aplikace schopná do určité míry předpovědět budoucí vývoj ceny akcie, popř. jiné datové řady. Tato práce ukázala, že není možné spoléhat se na technologie ve všech ohledech. Počítače dosud nemají dostatečný výpočetní výkon pro dokonalé předpovědi, ke kterým by bylo potřeba sledovat mnohem více faktorů, než je vývoj ceny v minulosti.

Dále byly otestovány čtyři různé způsoby, jak přistupovat k predikci cen. Jako nejspolehlivější typ předpovědi byl určen typ okamžité predikce všech hodnot. Cíl práce tedy byl naplněn. V rámci tohoto testu bylo také zjištěno nekalé chování některých internetových stránek (viz kapitola 4.2.2).

Práce může být následně rozšířena o sledování dějů výrazně ovlivňující chování cen, např. veřejné vystupování společností v médiích (např. uvedení nového produktu na trh nebo změny ve vedení).

7 ZDROJE

1. **Copeland, B. J.** Colossus | computer | Britannica. *Encyclopedia Britannica.* [Online] Encyclopædia Britannica, Inc., 11. 08 2014. [Citace: 01. 03 2020.] https://www.britannica.com/technology/Colossus-computer.

2. **Foote, Keith D.** A Brief History of Machine Learning. *DATAVERSITY.* [Online] DATAVERSITY Education, 26. 03 2019. [Citace: 01. 03 2020.] https://www.dataversity.net/a-brief-history-of-machine-learning/.

3. **Historian.** AI and Games Pioneer, A L Samuel. *i-programmer.info.* [Online] i-programmer.info, 24. 03 2015. [Citace: 01. 03 2020.] https://www.i-programmer.info/history/people/669-a-l-samuel-ai-and-games-pioneer.html.

4. **Klein, Daniel.** Mighty mouse - MIT Technology Review. *MIT Technology Review.* [Online] MIT Technology Review, 19. 12 2018. [Citace: 01. 03 2020.] https://www.technologyreview.com/s/612529/mighty-mouse/.

5. **Fogg, Andrew.** A History of Machine Learning and Deep Learning | Import.io. *Import.io - Data Extraction, Web Data, Web Harvesting, Data Preparation, Data Integration.* [Online] Import.io, 05. 06 2017. [Citace: 02. 03 2020.] https://www.import.io/post/history-of-deep-learning/.

6. **Blog.** What is Machine Learning? A definition - Expert System. *Expert System: Artificial Intelligence: Cognitive Computing Company.* [Online] Expert System S.p.A., 07. 03 2017. [Citace: 02. 03 2020.] https://expertsystem.com/machine-learning-definition/.

7. **Brownlee, Jason.** Supervised and Unsupervised Machine Learning Algorithms. *Machine Learning Mastery.* [Online] Machine Learning Mastery Pty. Ltd., 16. 03 2016. [Citace: 04. 03 2020.] https://machinelearningmastery.com/supervised-and-unsupervised-machine-learning-algorithms/.

8. **Guru99.** Reinforcement Learning: What is, Algorithms, Applications, Example. *Meet Guru99 - Free Training Tutorials & Video for IT Courses.* [Online] Guru99. [Citace: 02. 03 2020.] https://www.guru99.com/reinforcement-learning-tutorial.html.

9. **TIOBE.** index | TIOBE. *TIOBE - The Software Quality Company.* [Online] TIOBE Software BV. [Citace: 02. 03 2020.] https://www.tiobe.com/tiobe-index/.

10. **TensorFlow.** Why TensorFlow. *TensorFlow.* [Online] Google Brain Team. [Citace: 02. 03 2020.] https://www.tensorflow.org/about.

11. **Wikipedia.** Keras - Wikipedia. *Wikipedia, the free encyclopedia.* [Online] Wikimedia Foundation, 02. 02 2020. [Citace: 02. 03 2020.] https://en.wikipedia.org/wiki/Keras.

12. **scikit-learn.** About us — scikit-learn 0.22.1 documentation. *scikit-learn: machine learning in Python — scikit-learn 0.22.1 documentation.* [Online] scikit-learn developers. [Citace: 02. 03 2020.] https://scikit-learn.org/stable/about.html.

13. **Yahoo!** Yahoo Finance - Stock Market Live, Quotes, Business & Finance News. *Yahoo Finance.* [Online] Verizon Media. https://finance.yahoo.com/.

14. **Aroussi, Ran.** yfinance · PyPI. *PyPI · The Python Package Index.* [Online] Python Software Foundation. [Citace: 02. 03 2020.] https://pypi.org/project/yfinance/.

15. **pandas.** About pandas. *pandas - Python Data Analysis Library.* [Online] NumFOCUS. [Citace: 02. 03 2020.] https://pandas.pydata.org/about/index.html.

16. **Hunter, John D.** History — Matplotlib 3.1.3 documentation. *Matplotlib: Python plotting — Matplotlib 3.1.3 documentation.* [Online] Matplotlib, 2008. [Citace: 02. 03 2020.] https://matplotlib.org/3.1.3/users/history.html.

17. **Hochreiter, Sepp a Schmidhuber, Jürgen.** *LONG SHORT-TERM MEMORY.* München, Lugano : autor neznámý, 1997.

18. **Singh, Aishwarya.** Predicting the Stock Market Using Machine Learning and Deep Learning. *Analytics Vidhya.* [Online] Analytics Vidhya, 25. 10 2018. [Citace: 04. 03 2020.] https://www.analyticsvidhya.com/blog/2018/10/predicting-stock-price-machine-learningnd-deep-learning-techniques-python/.

19. **Rockikz, Abdou.** How to Predict Stock Prices in Python using TensorFlow 2 and Keras. *Python Code.* [Online] Python Code, 02 2020. [Citace: 04. 03 2020.] https://www.thepythoncode.com/article/stock-price-prediction-in-python-using-tensorflow-2-and-keras.

20. **colah.** Understanding LSTM Networks -- colah's blog. *colah's blog.* [Online] 27. 08 2015. [Citace: 02. 03 2020.] https://colah.github.io/posts/2015-08-Understanding-LSTMs/.

8 OBRÁZKY