EKONOMICKÁ UNIVERZITA V BRATISLAVE
FAKULTA HOSPODÁRSKEJ INFOMATIKY

Evidenčné číslo: 103004/B/2024/36145173622521348

PREHĽAD A POROVNANIE KNIŽNÍC STROJOVÉHO UČENIA V JAZYKU PYTHON

Bakalárska práca

2024 **Juraj Dzurenda**

EKONOMICKÁ UNIVERZITA V BRATISLAVE
FAKULTA HOSPODÁRSKEJ INFOMATIKY

PREHĽAD A POROVNANIE KNIŽNÍC STROJOVÉHO UČENIA V JAZYKU PYTHON

Bakalárska práca

Študijný program: Hospodárska informatika

Študijný odbor: Ekonómia a manažment

Školiace pracovisko: Katedra aplikovanej informatiky

Vedúci záverečnej práce: Ing. Erika Mináriková

Bratislava 2024 **Juraj Dzurenda**

Čestné vyhlásenie

Čestne vyhlasujem, že som bakalársku prácu s názvom: Prehľad a porovnanie knižníc strojového učenia v jazyku Python, vypracoval samostatne pod vedením svojej školiteľky, a že som uviedol všetku použitú literatúru.

....

Miesto, dátum Podpis študenta

Poďakovanie: Týmto spôsobom by som sa chcel poďakovať svojej vedúcej bakalárskej práce Ing. Erike Minárikovej za cenné rady a odporúčania, ktoré mi pomohli pri vypracovaní tejto práce.

ABSTRAKT

DZURENDA, Juraj: *Prehľad a porovnanie knižníc strojového učenia v jazyku Python*. – Ekonomická univerzita v Bratislave. Fakulta hospodárskej informatiky, Katedra aplikovanej informatiky. – Vedúci/vedúca záverečnej práce : Ing. Erika Mináriková – Bratislava: FHI EU, 2024, 55 s.

Záverečná práce je vypracovaná na tému „Prehľad a porovnanie knižníc strojového učenia v jazyku Python". Cieľom záverečnej práce je analyzovať a následne vytvoriť prehľad hlavných technických charakteristík knižníc strojového učenia v jazyku Python. Existuje mnoho knižníc strojového učenia v jazyku Python ako TensorFlow, PyTorch, Keras, Scikit-learn, SciPy. V práci vytvoríme prehľad týchto knižníc a na praktických príkladoch ich porovnáme.Jednotlivé časti záverečnej práce sú zamerané na vysvetlenie rozdielov medzi knižnicami určenými na strojové učenie a ich porovnanie, oboznámenie sa s problematikou strojového učenia v Pythone ako aj porozumenie jeho algoritmom. Výsledkom riešenia danej problematiky je porovnanie rozdielov výkonu a ďalších parametrov knižníc na rôznych príkladoch a datasetoch.

Kľúčové slová: Python, strojové učenie, neurónové siete, NumPy, SciPy, Jax, Scikit-learn, PyTorch, TensorFlow, Keras.

ABSTRACT

DZURENDA, Juraj: Overview and comparison of machine learning libraries in Python. – University of Economics in Bratislava. Faculty of Economic Informatics, Department of Applied Informatics–Supervisor of final thesis: Ing. Erika Mináriková–Bratislava: FHI EU, 2024,55p.

The final thesis is prepared on the topic „Overview and comparison of machine learning libraries in Python“. The goal was to analyze and produce an overview of main technical characteristics of machine learning libraries in Python. Lots of machine learning libraries for Python exist, such as TensorFlow, PyTorch,, Keras, Scikit-learn, SciPy. In thesis we will create an overview of these libraies and compare them on practical examples.Parts of this thesis were focused on explaining differences in among libraries created for machine learning and their comparison, getting to know the basics of machine learning Python and understanding its algorithms. The results of solving these problematics is comparison of performance of libraries and other parameters on different examples and datasets.

Keywords: Python, machine learning, neural networks, NumPy, SciPy, Jax, Scikit-learn, PyTorch, TensorFlow,Keras.

OBSAH

ÚVOD

Strojové učenie je súčasťou informatickej technológie známej pod názvom "Umelá inteligencia", ktorá sa stáva každým dnom populárnejšia vo svete okolo nás. Využitie pozorujeme napríklad v algoritmoch na rôznych stránkach ktoré nám odporúčajú rôzne možnosti vybrané na základe dát vyzbieraných od nás, generovanie textu, obrázkov, navigovanie a mnohých ďalších. Strojové učenie je podoblasťou takejto technológie a ponúka jej možnosť reagovať na rôzne vstupy bez toho aby na také vstupy bola naprogramovaná, dokáže odpovedať na základe naučených dát.

V našej práci sa sústredíme na porovnávanie knižníc jazyka Python, ktoré sú využívané na strojové učenie. Prvá kapitola používateľovi vysvetľuje základy problematiky strojového učenia a umožňuje porozumenie základov pre čitateľa, predstavuje možnosti aplikácie strojového učenia na rôzne príklady a uvádza rôzne metódy. Sú tu tiež opísane knižnice, ktoré práca obsahuje. Druhá kapitola popisuje ciele, ktoré by sme radi v našej práci dosiahli. V tretej kapitole vysvetľujeme ako budeme knižnice porovnávať podľa vybraných kritérií v teoretickej a praktickej časti. V tejto kapitole tiež popíšeme datasety Iris a Mnist, ktoré na porovnávanie v praktickej časti využijeme. Obsah štvrtej kapitoly je zameraný na výsledky práce. Na začiatku je teoretické porovnanie knižníc a neskôr prejdeme na analýzu získaných výsledkov pomocou spustenia algoritmov. Knižnice porovnávame na základe výkonu a iných vlastností knižníc a uvádzame závery, ku ktorým sme dospeli počas ich testovania. V závere sa sústredíme na diskusiu v ktorej ustanovujeme niekoľko možných ciest smeru ďalšieho výskumu.

1 SÚČASNÝ STAV RIEŠENEJ PROBLEMATIKY DOMA A V ZAHRANIČÍ

Téma umelej inteligencie je v dnešnej dobe veľmi populárna, je využívaná takmer všade okolo nás a to aj bez nášho vedomia. Súčasťou umelej inteligencie je aj strojové učenie (Machine learning), ktorého algoritmy sú základom umelej inteligencie.

1.1 Strojové učenie

Je častou umelej inteligencie, ktorá zodpovedá za jej schopnosť na základe analýzy dát získavať nove informácie, rozvíjať a zdokonaľovať sa, namiesto toho aby museli byť nové schopnosti vyslovene naprogramované. Zlepšovaním algoritmov strojového učenia a zvyšovaním výkonu počítačov sa spracovávajú obrovské množstvá dát vysoko efektívnym spôsobom a výsledky dosiahnuté umelou inteligenciou sa tým pádom zlepšujú [1].

1.1.1 Aplikácie využívajúce strojové učenie

Kým pred pár rokmi sme si pri vjazde na platené parkovisko museli zobrať lístok a pri výjazde ho zas vložiť do stojanu, dnešné moderné parkoviská majú systém schopný na základe rozpoznania ŠPZ pomocou kamery evidovať vozidlá a pri výjazde vyhodnotiť či a koľko má vodič zaplatiť. Systém umožňujúci rozpoznávanie ŠPZ vozidiel je založený na strojovom učení, kde sa algoritmus najprv učí pomocou analýzy veľkého množstva dát v podobe značiek vozidiel, a následne je schopný rozoznávať značky s veľkou presnosťou, ktorá sa dá ešte zdokonaliť ďalším učením [2].

Ďalšími oblasťami, kde sa strojové učenie využíva v aplikáciách umelej inteligencie sú algoritmy sociálnych sietí, reklamu aj príspevky, ktoré vidíme na sociálnych sieťach sú personalizované priamo pre každého konkrétneho užívateľa na základe toho, čo vyhľadáva na internete, aké príspevky sa mu páčia, zdieľa a komentuje. Algoritmus sa učí a počas každého kliku odporúča viac a viac personalizované príspevky aby používateľa udržal zaujatého čo najdlhšie[3].

Pri vytváraní prezentácií je umelá inteligencia schopná vytvoriť kompletnú prezentáciu pre používateľa za pár sekúnd, na základe poskytnutých materiálov alebo čisto len podľa zadanej témy.

Dôležitou oblasťou je aj vývoj nových liekov, kde farmaceutické firmy počas pandémie začali mať o umelú inteligenciu veľký záujem a dokážu ju využiť na vytvorenie nových liekov vo veľmi krátkom čase.

Ďalším príkladom odvetvia s implementovaním technológie umelej inteligencie je predpoveď počasia. Tradičné metódy predpovede počasia sú založene na riešení fyzikálnych modelov, ktoré vyžadujú veľké množstvo matematických výpočtov. V článku [4] sa tieto tradičné metódy nahradili algoritmami strojového učenia a dosiahli presnosť predpovede viac ako 84%.

Takýchto oblastí by sme mohli vymenovať ešte veľa.

1.1.2 Metódy strojového učenia

Poznáme tri základné skupiny metód strojového učenia [5]:

- Učenie s učiteľom (supervised learning),
- Učenie bez učiteľa (unsupervised learning),
- Učenie formou odmeňovania (reinforcement learning).

Učenie s učiteľom vyžaduje dostatočne veľké množstvo dát, ktoré obsahujú vstupné dáta a k nim správne priradene výstupné dáta. Algoritmus strojového učenia sa na týchto dátach „natrénuje“ tak, že keď potom dostane doteraz neznáme vstupné údaje, dokáže správne určiť výstupné dáta, ktoré im zodpovedajú. Typickým príkladom takéhoto učenia je klasifikácia, kedy algoritmy strojového učenia dokážu klasifikovať vstupný objekt do jednej z výstupných kategórii.

Učenie bez učiteľa sa požuvá vtedy, keď máme k dispozícii len vstupné dáta bez priradených výstupných dát. Algoritmy strojového učenia z tejto skupiny dokážu zoskupiť dáta do špecifických skupín podľa ich spoločných vlastností. Typickým príkladom takéhoto učenia je segmentácia zákazníkov, ktorá sa využije napríklad pri ponuke reklamy.

Učenie formou odmeňovania nevyžaduje vopred pripravené vstupné údaje na trénovanie. Namiesto toho sa v algoritme implementujú pravidlá „odmeňovania“ formou odmeňovacej funkcie, ktorá rozhoduje o výsledku. Aplikáciou tejto funkcie na jednotlivé prípady dokáže systém vždy vyberať možnosť, ktorá je najvýhodnejšia z pohľadu odmeny. Príkladom odmeňovania sú algoritmy použité v šachovej hre.

Niektoré zdroje uvádzajú aj ďalšie skupiny metód strojového učenia ako napríklad [6] uvádza aj učenie pod čiastočným dohľadom (semi-supervised learning), ktoré predstavuje kombináciu učenia s učiteľom a bez neho. Uvádza aj ďalšiu skupinu a to síce učenie samokontrolou (self-supervised learning), kde podobne ako pri učení bez učiteľa, sa použijú len vstupné dáta. Tieto metódy dokážu použiť časť vstupných údajov pre automatické generovanie chýbajúcich výstupných dát a takto transformujú učenie bez učiteľa na učenie s učiteľom.

V našej práci s budeme zaoberať len skupinou strojového učenia s učiteľom.

1.1.3 Úlohy strojového učenia s učiteľom

Podľa [7] sa uvádzajú dve základne úlohy pre strojové učenie s učiteľom.

Klasifikačná úloha (classfication task)

Klasifikačná úloha v strojovom učení rieši problém zaraďovania objektov do tried. Objekty majú priradené vlastnosti, ktoré sú vyjadrené v číselnej forme, pri čom každý objekt prislúcha pravé jednej triede. Úlohu je zistiť, do ktorej triedy patrí objekt s určitými vlastnosťami.

Počet vlastností objektov môže byť rôzny, sú datasety, kde majú objekty priradené 4 vlastnosti no aj datasety, kde má objekt viac ako 1000 vlastnosti, takéto objekty sú často pri spracovávaní obrázkov.

Príkladom pre klasifikačnú úlohu je rozpoznávanie dopravných značiek pri ceste. Dopravné značky sú objekty, ktoré sa klasifikujú. Objekty majú podobu obrázkov, ktoré môžu byť odfotené kamerou s vhodným rozlíšením. Každý pixel tejto fotografie predstavuje jednu vlastnosť, ktorá nadobudne hodnotu intenzity pixelu na fotografii, čo je číselná hodnota. V prípade, že kamera ma rozlíšenie 640x360 tento objekt ma 230400 vlastností. V tomto príklade triedy predstavujú jednotlivé značky, a úlohou klasifikácie je zaradiť obrázok do správnej triedy.

Regresná úloha (Regression task)

Regresná úloha je podobná klasifikačnej úlohe, rozdiel je, že kým klasifikačná úloha zaradovala objekt do triedy, regresná priradí k objektu výsledok v číselnej forme. Teda pri regresnej úlohe je počet tried nekonečný alebo príliš vysoký na to, aby sme ju mohli považovať za klasifikačnú úlohu.

Príkladom pre regresnú úlohu je predikcia ceny nehnuteľnosti. V tomto prípade je objekt (nehnuteľnosť) vyjadrená vlastnosťami ako je napríklad rozmer, počet izieb, rozloha miestnosti, vek nehnuteľnosti, lokalita a podobné. Výsledkom je číselná hodnota nehnuteľnosti.

V našej práci sa budeme zaoberať len klasifikačnými úlohami.

1.1.4 Algoritmy strojového učenia pre klasifikačnú úlohu

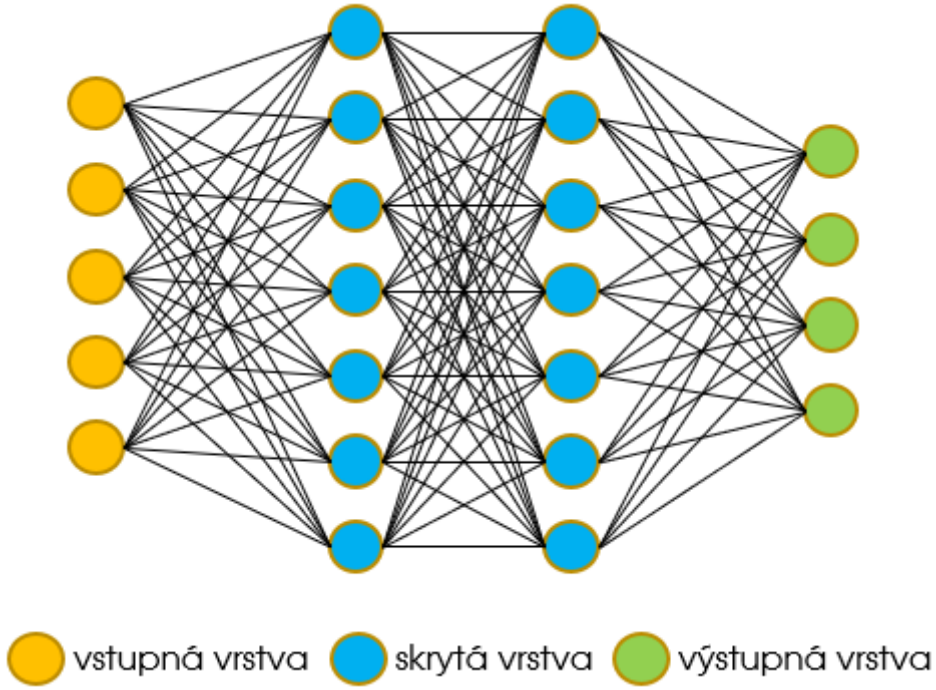
V práci [8] je popísaných 7 algoritmov, ktoré sa dajú použiť pri riešení klasifikačných úloh pomocou strojového učenia s učiteľom:

- Algoritmus pre hľadanie K-najbližších susedov (K-nearest neighbours),
- Lineárne modely (Linear models),
- Naive Bayes classifiers,
- Rozhodovacie stromy (Decision Trees),
- Ensembles of Decision Trees,
- Kernelized Support Vector Machines,
- Neurónové siete (Neural Networks (Deep learning)).

Najpopulárnejšie Pythonovské knižnice zamerané na strojové učenie používajú rôzne algoritmy ale niektoré z nich sú zamerané len na neurónové siete (Pytorch,TensorFlow, Keras) preto naše knižnice budeme porovnávať na riešení úloh pomocou tohto algoritmu.

Model Neurónovej siete bol inšpirovaný spôsobom prepojenia neurónov v ľudskom mozgu. Mozgové neuróny sa skladajú z tiel buniek a vodivých výbežkov (dendritov), ktorými sú poprepájané. Na základe prijatých signálov sa neurón môže rozhodnúť zareagovať a ďalším neurónom vyslať nový signál. Každý neurón reaguje na signály špecifickým spôsobom, jeho prispôsobovanie sa ma kľúčový význam pre funkciu ako ľudská pamäť a učenie[9].

V algoritmoch strojového učenia je neurónová sieť zložená z konečného počtu uzlov, ktoré nazývame tiež neuróny. Neuróny sú usporiadané do vrstiev – jedna vstupná vrstva, niekoľko skrytých vrstiev a jedna výstupná vrstva. Neuróny medzi vrstvami majú prepojenia, ktoré majú priradené číselné hodnoty nazývane váhy, ktoré sa počas učenia modelu upravujú. Množstvo neurónov a skrytých vrstiev sú parametre neurónovej siete, ktoré významne ovplyvňujú jej presnosť a rýchlosť učenia [10]. Štruktúra neurónovej siete s dvoma skrytými vrstvami je znázornená na obrázku 1.



Obr. 1: Model neurónovej siete. [10]

Pri riešení klasifikačných úloh strojového učenia sa počet uzlov na vstupnej vrstve rovná počtu vstupných vlastností analyzovaných objektov a počet výstupných vrstiev zodpovedá počtu kategórii, do ktorých klasifikujeme objekty. Neurónová sieť znázornená na obrázku 1 by sa dala použiť pre objekty, ktoré majú 5 vstupných vlastnosti a zaradujeme ich do jednej zo 4 kategórii.

Neurónová sieť sa učí takým spôsobom, že vstupné dáta prechádzajú sieťou smerom od vstupnej vrstvy k vrstve výstupnej. Jeden prechod všetkých vstupných dát sa nazýva epocha [11]. Počas prechodu vstupných (trénovacích) dát cez uzly siete sa váhy na prepojeniach uzlov prispôsobujú prechádzajúcim dátam, čím sa model neurónovej siete učí. Počet epoch je dôležitý parameter pri riešení úloh strojového učenia pomocou neurónových sietí pretože zvyšovanie tohto parametra väčšinou znamená síce pomalšie učenie, ale aj vyššiu presnosť modelu neurónovej siete.

Keď sa učenie ukončí môžeme model použiť pre zaradenie nového (testovacieho) objektu. V tomto prípade testovací objekt prechádza uzlami neurónovej siete, ktorá ho s pomocou nastavených váh medzi prepojenými uzlami zaradí do jednej z kategórií vo výstupnej vrstve.

1.2 Programovacie jazyky používané pre strojové učenie

Podľa [12] knižnice strojového učenia sú s k dispozícii v jazykoch Python ,R ,C++ , Java a JavaScript, pričom voľbou číslo jedna pre mnohých programátorov je Python.

Jazyk Python je druhý najpopulárnejší jazyk programátorov na GitHube [13]. Dôvodom jeho popularity je jeho jednoduchosť a ľahko porozumiteľný syntax.

Python je jeden z najpoužívanejších a najrozšírenejších programovacích jazykov dnešnej doby, je jednoduchý na používanie vďaka čomu je uprednostňovaný medzi začiatočníkmi. Vznikol v roku 1991 a v roku 2008 vznikla jeho verzia 3. V súčasnosti aktuálna verzia je 3.12 z októbra 2023 [14]. Je to univerzálny objektovo orientovaný jazyk podobne ako C++, C# alebo Java. Jeho využitie je rôznorodé, slúži na vytváranie softwaru, automatické testovanie ale aj na analýzu dát prípadne pomoc s matematickými úlohami.

Pre machine learning je využívaný taktiež preto, že v strojovom učení je potrebná práce s dátami a taktiež je tu veľmi potrebná matematika, na čo ma Python veľmi dobré a ľahko dostupné knižnice, ktoré sú zadarmo.

Ďalším vhodným jazykom na riešenie úloh strojového učenia je jazyk R. R je programovací jazyk, ktorý je určený predovšetkým na štatistiku, matematiku a grafické spracovanie dát. V roku 1993 bol spustený profesorom Ross Ihakom a Robertom Gentelmanom za účelom vyučovania štatistiky na univerzite .V strojovom učení sa dá využiť na rôzne veci a jeho veľkou výhodou je jeho schopnosť lepšieho využitia štatistických metód.

Jazyk C++ sa niekedy používa v strojovom učení kvôli jeho rýchlosti ale podobne ako Java a JavaScript, počet knižníc pre strojové učenie je obmedzený.

Podľa [15] je pre strojové učenie najviac využívaný jazyk Python, využíva ho viac ako 65% opýtaných programátorov venujúcich sa strojovému učeniu.

1.3 Knižnice pre strojové učenie

Knižnice sú nástrojom, ktorý umožňuje vyriešiť problémy a úlohy strojového učenia aj ľudom, ktorí nie sú experti pre algoritmy, keďže potrebné algoritmy sú dodávane knižnicami.

Niektoré knižnice pomáhajú pri analýze a transformácii vstupných dát iné poskytujú matematické algoritmy, ďalšie pomáhajú pri vizualizácii údajov a iné implementujú jednotlivé algoritmy strojového učenia

V našej práci sa zameriame na knižnice jazyka Python, ktoré implementujú algoritmy neurónových sietí. K nim pridáme tri knižnice, ktoré implementujú matematické algoritmy, metódy na prípravu dát a analýzu výsledkov. Jednotlivé knižnice si popíšeme v nasledujúcich podkapitolách.

1.3.1 NumPy

Je to knižnica vytvorená v roku 2005, vytvoril ju Travis Oliphan. NumPy je skratka pre Numerical Python (Číselný Python - doslovný preklad). Nie je to tak úplne knižnica pre strojové učenie ale je v takýchto projektoch široko využívaná ostatnými knižnicami na vykonávanie matematických operácií a prípravu dát. Jedna z najpopulárnejších knižníc v Pythone, používaná v mnohých projektoch v oblasti IT, akademického vzdelávania, a matematiky, umožňuje aj vizualizáciu ako kreslenie grafov a priamok.

Domáca stránka knižnice:

https://numpy.org/

Dokumentácia:

Knižnica je dôkladne popísaná priamo na jej domácej stránke v sekcii Documentation. Je tu prehľadne napísaný návod na začiatok vysvetľujúci aj jej špeciálnu funkciu NumPy array, pomocou ktorej zjednodušuje základné polia v Pythone. Tento návod obsahuje aj príklady kódu. Okrem toho sú tu aj komplikovanejšie návody aj návod, ktorý vysvetľuje užívateľom ako knižnicu rozšíriť o svoj vlastný kód.

Použitie knižnice:

Knižnica poskytuje štruktúry na uchovávanie dát a matematické operácie vhodné pre riešenie úloh strojového učenia.

Inštalácia knižnice:

pip3 install numpy

Závislosti knižnice:

Žiadne (0 knižníc)

Licenčný model:

Open-source BSD licencia

1.3.2 SciPy

Bola vytvorená už v roku 2001, za jej vznikom je tiež Travis Oliphan, ale dnes slúži skôr ako nadstavba knižnice NumPy, čo jej umožňuje prepoužívať jej funkcie. Knižnica poskytuje aj zložitejšie funkcie, čo umožňuje riešenie komplikovanejších úloh. Je jednoduchá na použitie aj pre nových užívateľov. Umožňuje riešenie matematiky, štatistiky, optimalizáciu a ďalšie.

Domáca stránka knižnice:

https://scipy.org/

Dokumentácia:

Knižnica je dôkladne popísaná priamo na jej domácej stránke v sekcii Documentation. Je tu prehľadne napísaný tutoriál, ktorý obsahuje aj príklady kódu. Okrem toho sú tu aj komplikovanejšie návody ako aj návod, ktorý užívateľom vysvetľuje ako knižnicu rozšíriť o svoj vlastný kód.

Inštalácia knižnice:

pip3 install scipy

Závislosti knižnice:

numpy (1 kniznica)

Licenčný model:

Open-source BSD licencia

1.3.3 Jax

Knižnica vytvorená pre vysoko výkonné počítanie numerických výpočtov, ako aj rozsiahle úlohy strojového učenia. Má možnosť aj vizualizácie a kreslenia grafov. Knižnice sú si veľmi podobné s knižnicou NumPy, Jax ma z nej prebraté schopnosti ako tvorbu špeciálnych polí. Jax vie pracovať priamo s poľom vytvoreným pomocou Numpy.

Domáca stránka knižnice:

https://jax.readthedocs.io/en/latest/index.html

Dokumentácia:

Na stránke majú dokumentáciu v sekcii Developer docs a tutoriály v sekciách Getting stared a User Guides.

Inštalácia knižnice:

pip3 install "jax[cpu]"

Závislosti knižnice:

jaxlib, ml-dtypes, opt-einsum, numpy, scipy (5 knižníc)

Licenčný model:

Open-source Apache 2.0 licencia

1.4 Knižnice podporujúce strojové učenie

1.4.1 Scikit-learn

Knižnica vznikla v roku 2007 ako projekt Davida Cournapeau. Implementuje rôzne algoritmy strojového učenia, medzi nimi aj algoritmy neurónových sietí, je predurčená skôr na menšie projekty a umožňuje aj štatistické modelovanie. Dnes je rozvíjaná viac komunitou.

Domáca stránka knižnice:

https://scikit-learn.org/

Dokumentácia:

Popísaná priamo na ich stránke v sekciách API, User Guide, Examples a More -> Tutorials. Nachádzajú sa tu rôzne návody na písanie kódu pre rôzne úlohy strojového učenia, vrátane funkčných príkladov.

Inštalácia knižnice:

pip3 install scikit-learn

Závislosti knižnice:

threadpoolctl, joblib, numpy, scipy (4 knižnice)

Licenčný model:

Open-source BSD-3 licencia

1.4.2 PyTorch

Pôvodne vytvorená spoločnosťou Meta AI v roku 2016, ktorá ju využíva na algoritmy svojich sociálnych sietí. Knižnica je optimalizovaná pre deep learning, implementuje rôzne komplexné algoritmy neurónových sieti. Dá sa použiť na riešenie matematických úloh aj na riešenie úloh strojového učenia, podporuje spúšťanie výpočtov na CPU aj GPU.

Domáca stránka knižnice:

https://pytorch.org/

Dokumentácia:

Popísaná priamo na ich stránke v sekcii Docs na stránke sa nachádzajú aj tutoriály pre začiatočníkov. Dostupné sú aj videá, a na internete pre ňu existuje veľa návodov z tretích strán.

Inštalácia knižnice:

pip3 install torch

Závislosti knižnice:

mpmath, typing-extensions, sympy, networkx, MarkupSafe, fsspec, filelock, jinja2

(8 knižníc)

Licenčný model:

Open-source BSD-3 licencia

1.4.3 TensorFlow

Vytvorená v roku 2015 a udržiavaná spoločnosťou Google, postupne sa stala jednou z najpoužívanejších Python knižníc na svete v oblasti strojového učenia a deep learningu. Je vhodné ju využívať pre rozpoznávanie obrázkov a znakov, nakoľko má dobre rozpracované neurónové siete. Jej meno TensorFlow obsahuje slovo Tensor čo je označenie technológie

na skladovanie číselných dát, technológia, ktorú aj využíva. Knižnica poskytuje aj metódy pre riešenie matematických úloh s dátami typu Tensor. Knižnica umožňuje prácu s dátami na CPU aj GPU.

Domáca stránka knižnice:

tensorflow.org

Dokumentácia:

Je dôkladne spísaná na ich stránke v sekcii Learn, je tam klasický návod pre začiatočníkov, v ktorom je popísaných niekoľko princípov ako je návod pre skúsených užívateľov. Stránka ponuka aj náučne videá.

Inštalácia knižnice:

pip3 install tensorflow

Závislosti knižnice:

namex, libclang, flatbuffers, wrapt, wheel, werkzeug, urllib3, termcolor, tensorboard-data-server, six, setuptools, pygments, protobuf, packaging, optree, opt-einsum, ml-dtypes, mdurl, markdown, idna, h5py, grpcio, gast, charset-normalizer, certifi, absl-py, tensorboard, requests, markdown-it-py, google-pasta, astunparse, rich, keras, tensorflow-intel, typing-extensions, numpy, MarkupSafe (37 knižníc)

Licenčný model:

Open-source Apache 2.0 licencia

1.4.4 Keras

Funguje ako nadstavba pre knižnice Jax, TensorFlow a PyTorch ktoré Keras používa na samotný výpočet. Jeho účelom je zjednodušiť používanie týchto základných knižníc. Projekt bol zverejnený v roku 2015 a za jeho vývojom je spoločnosť Google.

Všetky 3 knižnice sa s knižnicou Keras používajú jednotným spôsobom, voľba knižníc sa realizuje len nastavením premennej operačného systému KERAS_BACKEND, ktorá môže nadobúdať jednu z hodnôt „jax“, „torch“ alebo „tensorflow“.

Zaujímavosťou je, že knižnica TensorFlow ma závislosť na knižnicu Keras, to znamená, že ju inštaluje ako svoju súčasť a na vytvorenie modelu neurónovej siete ako aj jeho použitie používa tie isté príkazy ako knižnica Keras.

Domáca stránka knižnice:

https://keras.io/

Dokumentácia:

Zo všetkých knižníc je aktuálne Keras asi najlepšie zdokumentovaná, na ich stránke majú v sekcii Guide návody na rôzne využitia ich knižnice doplnené o príklady kódu.

Inštalácia knižnice:

pip3 install keras

Závislosti knižnice:

absl-py, numpy, rich, namex, h5py, optree, ml-dtypes, typing-extensions, markdown-it-py, pygments, mdurl (11 knižníc)

Licenčný model:

Open-source Apache 2.0 licencia

2 CIEĽ PRÁCE

Cieľom práce je analyzovať a následne vytvoriť prehľad hlavných technických charakteristík knižníc strojového učenia v jazyku Python.

Existuje mnoho knižníc strojového učenia v jazyku Python ako TensorFlow, PyTorch, Keras, Scikit-learn, SciPy. V práci vytvoríme prehľad týchto knižníc a na praktických príkladoch ich porovnáme

Cieľom práce je porovnať rôzne knižnice jazyku Pythonu, ktoré sú využívané pre algoritmy strojového učenia. Čitateľ by mal nabrať všeobecný prehlaď o téme strojové učenie a výstupom práce sú poznatky usmerňujúce užívateľa k výberu najvhodnejšej knižnice pre projekt. Tiež ukážeme ako vyzerá kód písaný s využitím knižnice a ako ich implementovať, ako aj to, aké knižnice sa pre strojové učenie využívajú.

Téme sa venujeme kvôli tomu, že téma umelej inteligencie posledné roky naberá na popularite, tým pádom sa dostáva do popredia aj téma strojového učenia, ktorá s ňou úzko súvisí. Je teda potrebné aby ľudia pohybujúci sa v IT mali o téme prehlaď. Jej využitie takmer nepozná hraníc a ma potenciál ovplyvniť životy ľudí do značnej miery, čoho sa už aj dopúšťa, často bez nášho vedomia.

V práci sa budeme venovať porovnávaniu nielen knižníc, ktoré priamo využívajú algoritmy strojového učenia ale aj knižniciam, ktoré umožňujú tvorbu takýchto algoritmov a to aj preto lebo strojové učenie a umelá inteligencia je do veľkej miery založená na matematike.

Náš ciel sa pokúsime dosiahnuť popísaním knižníc v teoretickej časti, následne sa však ponoríme hlboko do praktického využitia, keď si v praktickej časti ukážeme priamo funkčné algoritmy a analyzujeme výsledky ich práce po ich aplikovaní na dvoch datasetoch odlišnej veľkosti. Ukážeme aj to, že niektoré knižnice dosahujú rôzne výsledky na rôznych typoch projektov, niektoré môžu by vhodné na menšie a niektoré na väčšie projekty.

3 METODIKA PRÁCE A METÓDY SKÚMANIA

V tejto kapitole popíšeme ako sme porovnávali navzájom jednotlivé knižnice na riešení konkrétnych príkladov. Podrobne vysvetlíme príklady na ktorých sa porovnávajú knižnice aj metodiku porovnávania výsledkov.

Pythonovské knižnice sme porovnali z týchto hľadísk:

- Oblasť použitia knižnice,
- Dokumentácia knižnice,
- Závislosti knižnice na iných knižniciach,
- Typ licencie,
- Rýchlosť výpočtu knižnice,
- Presnosť výpočtu knižnice,
- Obtiažnosť použitia knižnice.

Prvé štyri hľadiska budeme porovnávať na základe informácii dostupných na domovských stránkach jednotlivých knižníc.

Posledné tri kritériá budeme porovnávať na troch praktických úlohách:

- Úloha I. - Riešenie sústavy lineárnych rovníc a inverznej matice,
- Úloha II. - Riešenie klasifikácie kvetov pomocou neurónovej siete (Dataset Iris),
- Úloha III. - Riešenie klasifikácie rukou písaných číslic pomocou neurónovej siete (Dataset Mnist).

Na úlohe I porovnáme matematické knižnice NumPy, SciPy, Jax a knižnice strojového učenia PyTorch a TensorFlow, ktoré tiež ponúkajú metódy pre riešenie matematických úloh.

Na úlohách II a III porovnáme knižnice strojového učenia TensorFlow,PyTorch, Scikit-learn a Keras.

Knižnice sme porovnávali praktickým výpočtom príslušných úloh. Výpočty prebehli na počítači s nasledovným HW vybavením:

- CPU-Procesor : I7-4790K,
- RAM:16GB,
- HDD: 249GB SSD,

- OS: WIN 10 Professional 64bit.

Výpočet prebiehal v Python s verziou 3.12.0 v prostredí Python IDLE. Okrem tohto prostredia nebol v čase vykonávania porovnania spustený žiaden iný užívateľsky program, čím sme obmedzili vplyv iných programov na výsledky porovnania.

Vplyv procesov počítača bežiacich na pozadí sa nepodarilo úplne obmedziť a čas výpočtu jednotlivých úloh nebol rovnaký a to ani v prípade že vstupné dáta boli rovnaké. Preto sme každý výpočet opakovali 5-krát a do výsledku porovnania sme zobrali aritmetický priemer týchto piatich výpočtov.

Niektoré knižnice počas výpočtov hlásili upozornenia ale výpočet vždy pokračoval ďalej. Tieto upozornenia sme nebrali do úvahy pretože doležíte bolo, že knižnica poskytla výsledok.

3.1 Metodika úlohy I. – Riešenie sústavy lineárnych rovníc a inverznej matice

Úlohy strojového učenia sú založené na riešení komplexných matematických úloh, a preto knižnice ktoré riešia úlohy strojového učenia implementujú matematické algoritmy buď priamo alebo používajú iné knižnice špecializované na matematické výpočty.

Ako je uvedené v predchádzajúcej kapitole, z knižníc ktoré sa používajú v oblasti strojového učenia podporuje matematické výpočty 5 knižníc:

- NumPy,
- SciPy,
- Jax,
- PyTorch,
- TensorFlow.

Tieto knižnice porovnáme na riešení dvoch matematických úloh – na riešení sústavy lineárnych rovníc a na výpočte inverznej matice.

Riešiť sústavu lineárnych rovníc:

$$\mathbf{A.x = b} \tag{1}$$

znamená nájsť vektor **x** ktorý spĺňa uvedenú rovnicu. Z matematiky je známe, že v prípade ak matica **A** je štvorcová a je regulárna, táto sústava ma jediné riešenie.

V našom experimente budeme porovnávať 5 pythonovských knižníc tak, že nájdeme riešenie takejto sústavy rovníc a zaznamenáme čas potrebný na výpočet. Postupne budeme riešiť sústavy rovníc rozmerov 10, 100, 500, 1000, 2500, 5000, 7500, 10000 a 15000.

Pre vytvorenie matice A sme pre jednotlivé rozmery matice (dim) použili príkazy knižnice NumPy:

```
A = -0.5 + numpy.random.rand(dim, dim)

numpy.fill_diagonal(A, 2*dim)
```

Matica **A** nadobúda náhode vygenerované hodnoty v intervale <-0.5;0.5) len na hlavnej diagonále sú hodnoty 2x rozmer matice. Teda ak je rozmer matice 10x10 tak na hlavnej diagonále budú čisla 20. Týmto sa zabezpečí že matica **A** je regulárna, to znamená že knižnice môžu hľadať riešenie sústavy rovníc.

Vektor pravej strany **b** sa vytvoril pomocou príkazu knižnice NumPy:

```
b = 3 + numpy.random.rand(dim)
```

Hodnoty vektora **b** sú náhodne generované čísla v intervale <3;4).

Knižnice sme porovnali z pohľadu výpočtového času. Tento čas sa získa tak, že zistíme aktuálny čas počítača tesne pred sledovaným výpočtom a ďalší čas na konci sledovaného výpočtu, následne vykonáme rozdiel časov. Na to používame príkazy jazyka Python:

```
t_start = time.time() * 1000

...sledovana operacia

t_end = time.time() * 1000
t_cas_operacie = t_end - t_start
```

Takto získaný čas ja čas potrebný na vykonanie sledovanej operácie v milisekundách.

Niektoré knižnice (PyTorch a TensorFlow) vyžadujú maticu A a vektor b v tvare tenzoru, čo je vnútorná dátová štruktúra, ktorú tieto knižnice používajú. Preto bola potrebná transformácia vygenerovaných dát do potrebného formátu. Čas potrebný na tieto transformácie sa nezarátaval do času výpočtu pre porovnanie knižníc.

Po každom výpočte sa zisťovala presnosť výpočtu. Presnosť sme zistili tak, že sme vypočítali rozdiel **Ax-b**, potom sa urobila absolútna hodnota každého prvku vypočítaného

vektora a do chyby výpočtu sa zobral najväčší prvok. Pre tento výpočet sme použili príkaz knižnice NumPy

```
err_numpy_eq = abs(max((numpy.dot(A, x_numpy) - b).flat,
key=abs))
```

Knižnice sme porovnali aj na úlohe výpočtu inverznej matice, to znamená, že pre maticu **A** sme hľadali maticu **Ainv** pre ktorú platí:

$$\mathbf{A*Ainv=E} \tag{2}$$

kde E je matica, ktorá obsahuje samé nuly len na hlavnej diagonále má jednotky. Z matematiky vieme že matica Ainv sa vždy dá nájsť keď je matica A štvorcová a regulárna. Maticu **A** sme pre túto úlohu negenerovali ale použili sme už vygenerované matice, ktoré boli použite pre výpočet sústavy lineárnych rovníc. Takto sme inverznú maticu hľadali postupne pre matice veľkosti 10, 100, 500, 1000, 2500, 5000, 7500, 10000 a 15000.

Čas výpočtu sme získali podobne ako pre výpočet sústavy rovníc a to tak, že sme zistili čas počítača pred výpočtom inverznej matice, potom čas počítača po výpočte a tieto časy sme odčítali.

Po každom výpočte sa zisťovala chyba nájdeného riešenia tak, že sa vypočítal rozdiel A*Ainv-E, všetky prvky výslednej matice sa dali do absolútnej hodnoty a vybral sa z nich najväčší prvok. Výpočet sa urobil s pomocou príkazu knižnice NumPy:

```
err_numpy_inv = abs(max((numpy.dot(A, Ainv_numpy) -
                         numpy.identity(dim)).flat, key=abs))
```

Obtiažnosť použitia knižníc pre výpočet sústavy rovníc aj inverznej matice sme porovnali na základe príkazov potrebných na výpočet v jednotlivých knižniciach.

Pre Knižnice PyTorch a TensorFlow sa vykonáva aj transformácia dát do tenzorov, čo však nie je nevýhodou týchto knižníc, pretože ak by vstupné dáta boli vytvárane vo forme tensorov tak by naopak knižnice Numpy a SciPy vyžadovali transformáciu týchto dát do ich štruktúr, preto takéto transformácie dátových typov nebudeme brať do úvahy pri vyhodnocovaní času ani pri hodnotení obtiažnosti použitia knižníc.

3.2 Metodika úlohy II. – Riešenie klasifikácie kvetov pomocou neurónovej siete

V tejto časti budeme porovnávať knižnice jazyka Python na praktickej úlohe klasifikácie kvetov kosatca s pomocou neurónovej siete.

Ako je uvedené v predchádzajúcej kapitole, z knižníc, ktoré sa používajú v oblasti strojového učenia podporujú algoritmy neurónových sietí nasledujúce knižnice:

- Scikit-learn,
- PyTorch,
- TensorFlow,
- Keras + Jax,
- Keras + PyTorch,
- Keras + TensorFlow.

Tieto knižnice porovnáme na klasifikácii objektov z datasetu Iris, pričom budeme vyhodnocovať ich rýchlosť, presnosť kategorizácie a tiež jednoduchosť implementácie.

Iris je jeden z najstarších datasetov pre strojové učenie používaný na porovnanie klasifikačných metód, je pomenovaný podľa kvetu na Slovensku známom ako kosatec. Dataset vznikol v roku 1936 a obsahuje 150 údajov o kvetoch kosatcov troch druhov Iris-setosa, Iris-versicolor, Iris-virginica. V datasete je 50 údajov o Iris-setosa, nasleduje 50 údajov o Iris-versicolor a na konci je 50 údajov o Iris-virginica [16]. Dataset sme získali zo zdroja [17].

Dataset má 4 vstupné vlastnosti:

- Dĺžka kališného lístka (sepal length),
- Šírka kališného lístka (sepal width),
- Dĺžka okvetného lístka (petal length),
- Šírka okvetného lístka (petal width).

ktoré sú uvádzane v dátach v centimetroch.

Obr. 2: Príklad objektov datasetu Iris. [16]

Objekty v datasete majú 3 kategórie, ktoré predstavujú 3 druhy kosatcov:

- Iris-setosa,
- Iris-versicolor,
- Iris-virginica.

Jeden riadok datasetu (napríklad riadok číslo 3) má tvar:

4.7,3.2,1.3,0.2,Iris-setosa

Kde na začiatku sú 4 vstupné údaje a na konci je trieda objektu. Tento riadok zodpovedá druhu kosatca Iris-setosa, pričom má dĺžku a šírku kališného lístka 4.9 cm a 2.5 cm, šírku a dĺžku okvetného lístka má 4.5 cm a 1.7 cm.

Dataset Iris sme pre účely tejto úlohy rozdelili na dve časti – trénovaciu časť a testovaciu časť. Keďže objekty v datasete sú zoradené podľa druhu kvetov, najprv sa dáta v datasete náhodne premiešali a potom sa celý dataset rozdelil na časť trénovacíach a testovacích dát. V trénovacej časti bolo cca 75% údajov (112 objektov) a v testovacej zvyšok (38 objektov). Rozdelenie dát sme urobili pomocou príkazu z knižnice Scikit-learn:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    train_size=0.75)
```

Knižnice sme porovnali tak, že sme v každej z nich vytvorili model neurónovej siete a tento model sme s pomocou trénovacích dát naučili rozpoznávať druhy kosatcov. Keď sa proces učenia ukončil, použili sme model pre predikciu tried tak pre testovacie ako aj pre trénovacie objekty.

Pre porovnávanie knižníc bol vytvorený model neurónovej siete s parametrami (hodnoty parametrov boli prevzaté z [18]):

- Počet uzlov vo vstupnej vrstve: 4,
- Počet skrytých vrstiev: 2,
- Počet uzlov v prvej skrytej vrstve: 25,
- Počet uzlov v druhej skrytej vrstve: 30,
- Počet uzlov vo výstupnej vrstve: 3,
- Metoda optimalizácie : Adam,
- Úroveň učenia: 0.01,
- Počet epoch: 10, 20, 50, 100, 250, 500, 750, 1000.

Ako je naznačené v parametri počet epoch, model bol vytvorený postupne pre 8 rôznych hodnôt počtu epoch nakoľko výpočet pre hodnotu 100 bol veľmi rýchly a neumožnil porovnanie knižníc.

Každý výpočet sme opakovali 5 x a do výsledku sme zobrali aritmetický priemer týchto meraní aby sme odstránili vplyv bežiacich procesov na pozadí počas počítača na výsledky porovnania.

Počas riešenia úlohy sme merali čas potrebný na učenie modelu ako je čas potrebný na predikciu tried trénovacích a testovacích objektov. Metodika merania času bola rovnaká ako v predošlých úlohách, to znamená, že s pomocou aktuálneho času počítača odmeraného pred a o vykonaní príslušnej operácie sme získali čas jej realizácie.

Presnosť modelu sme vyhodnocovali tak, že sme porovnali hodnoty predikované modelom so skutočnými hodnotami z datasetu a zobrali sme z nich aritmetický priemer. Výpočet sme urobili pre testovacie ako aj trénovanie dáta s pomocou knižnice NumPy príkazom:

```
score_test = numpy.mean(y_predict_test == y_test)
score_train = numpy.mean(y_predict_train == y_train)
```

Obtiažnosť použitia knižníc ohodnotíme na základe zložitosti kódu a jednoduchosti zostrojenia funkčného programu, ktorý rieši klasifikáciu objektov datasetu iris.

3.3 Metodika úlohy III. – Riešenie klasifikácie rukou písaných číslic pomocou neurónovej siete

Podobne ako v predošlej časti, budeme porovnávať knižnice jazyka Python na praktickom výpočte klasifikácie objektov, tento krát z datasetu Mnist.

Porovnanie urobíme s tými istými knižnicami, ktoré sme použili v predchádzajúcej kapitole:

- Scikit-learn,
- PyTorch,
- TensorFlow,
- Keras (Jax),
- Keras (PyTorch),
- Keras (TensorFlow).

Podobne ako s datasetom Iris, aj tu budeme knižnice porovnávať na klasifikácii objektov z datasetu Mnist, pričom budeme vyhodnocovať jeho rýchlosť, presnosť kategorizácie a tiež jednoduchosť implementácie.

Mnist je populárny dataset, ktorý obsahuje obrázky rukou písaných číslic od 0 po 9. Jeho názov je skratka od Modified National Institute of Standards and Technology, kde tento dataset vznikol. Často sa používa ako testovací dataset pre začínajúce projekty v oblasti strojového učenia pre oblasť rozpoznávania obrázkov.Podobne ako dataset Iris, aj dataset Mnist je verejne prístupný, my sme ho získal tiež z portálu OpenML [19].

Dataset obsahuje údaje o 70 000 obrázkoch, pričom každý obrázok má rozmery 28x28 bodov (pixelov). Obrázky sú čiernobiele, kde hodnota pixelu predstavuje intenzitu farby. Hodnota 0 znamená bielu farbu, hodnota 255 čiernu farbu a hodnoty medzi nimi znamenajú úroveň šedej farby príslušnej intenzity [20]. Jeden riadok v datasete ma 785 riadkov, pričom prvých 783 číslic predstavuje pixely obrázku 28x28 s hodnotami od 0 po 255. Posledné číslo predstavuje číslicu, ktorá je zobrazená na obrázku. Preto má dataset 783 vstupných vlastnosti a obrázky majú 10 kategórii, ktoré sú číslice od 0 po 9.

Na obrázku 3 sa nachádza prvých 100 záznamov datasetu Mnist.

Obr. 3: Prvých 100 obrázkov datasetu Mnist. [21]

Na rozdiel od datasetu Iris, údaje o obrázkoch v datasete Mnist sú náhodne usporiadané preto ich netreba miešať. Dáta sme rozdelili na časť trénovaciu a časť testovaciu tak, že do trénovacej časti sa zobralo prvých 60 000 údajov a zvyšok (10 000 údajov) boli testovacie dáta. Podobne ako v predošlej kapitole, aj pri klasifikovaní obrázkov dataset Mnist sme vytvorili model neurónovej siete pre všetky porovnávané knižnice. Po ukončení procesu učenia bol model použitý na identifikovanie obrázkov tak z testovacieho ako aj trénovacieho datasetu.

Čas učenia ako aj čas identifikovania obrázkov z oboch častí dát bol odmeraný a použitý pre porovnanie knižníc. Tiež sa vyhodnocovala presnosť knižníc porovnaním

klasifikovaných obrázkov so skutočnými hodnotami z datasetu, takým spôsobom ako s datasetom Iris. Aj spôsob porovnania obtiažnosti implementácie bol rovnaký.

Pre porovnanie knižníc bol vytvorený model neurónovej siete s parametrami (hodnoty boli prevzaté z [22]):

- Počet uzlov vo vstupnej vrstve: 783,
- Počet skrytých vrstiev: 1,
- Počet uzlov v prvej skrytej vrstve: 50,
- Počet uzlov vo výstupnej vrstve: 10,
- Metoda optimalizácie : SGD,
- Úroveň učenia: 0.1,
- Počet epoch: 10, 50, 100, 200, 300, 400, 500.

Aj v tejto úlohe budeme porovnávať knižnice pre rôzny počet epoch. Kým v datasete Iris sme mali 8 hodnôt pre epochy s najväčšou hodnotou 10 000, tu je 7 hodnôt s najväčšou hodnotou 500. Je to kvôli veľkosti datasetu Mnist, ktorý obsahuje 70 000 dát v porovnaní so 150 dátami Iris, čo veľmi predlžuje čas výpočtov.

Každý výpočet sa opakoval 5 x a do výsledku sme zobrali aritmetický priemer týchto meraní aby sme odstránili vplyv bežiacich procesov na pozadí počas počítania na výsledky porovnania. Podobne ako pri riešení úlohy s datasetom Iris, aj tu sa meral čas potrebný na učenie modelu a čas potrebný na klasifikáciu dát z testovacej aj trénovacej časti. Metodika merania času bola rovnako založená na zisťovaní aktuálneho času počítača pred a po vykonaní príslušných príkazov v programe. Aj presnosť modelu sa vyhodnocovala rovnako. Porovnali sme výsledné hodnoty klasifikácie obrázkov číslic získané pomocou neurónovej siete s tými ktoré boli v datasete Mnist a za výsledok sa zobral aritmetický priemer.

Obtiažnosť použitia knižníc bola opäť vyhodnotená na základe zložitosti kódu a jednoduchosti zostrojenia funkčného programu, ktorý rieši klasifikáciu objektov datasetu Mnist.

4. VÝSLEDKY PRÁCE

V tejto kapitole sa detailne zaoberáme výsledkami našej práce a posudzujeme, ktoré knižnice sa javia ako najvhodnejšie pre rozličné úlohy. Závery vyvodíme z analýzy dosiahnutých výsledkov.

4.1 Výsledky teoretického porovnania knižníc

4.1.1 Oblasť použitia knižníc

Na riešenie matematických úloh sa dajú použiť knižnice NumPy, SciPy, Jax, PyTorch a TensorFlow.

Na riešenie úloh strojového učenia sa dajú použiť knižnice Scikit-learn, PyTorch, TensorFlow a Keras pri čom Keras sa používa v kombinácii s Jax, PyTorch alebo TesorFlow. Z týchto knižníc Scikit-learn implementuje rôzne algoritmy strojového učenia a ostatné knižnice implementujú len algoritmy neurónových sieti.

4.1.2 Dokumentácia knižníc

Všetky knižnice majú dokumentáciu prístupnú na domovskej stránke. V dokumentácii sú opísane jednak metódy ktoré knižnice poskytujú ako aj stručne návody na ich použitie.

4.1.3 Závislosti knižníc na iných knižniciach

Niektoré knižnice potrebujú ďalšie knižnice pre svoje fungovanie. Knižnica NumPy nemá žiadne závislosti. SciPy ma jedinú závislosť, ktorou je knižnica NumPy. Knižnice Jax, PyTorch a Scikit-learn sú z pohľadu závislosti prevažne rovnaké, vyžadujú 4 - 8 ďalších knižníc. TensorFlow vyžaduje až 37 knižníc, to je zo všetkých najviac. Keras samotný vyžaduje 11 knižníc ale dá sa použiť len spolu s ďalšími knižnicami Jax, PyTorch, TensorFlow. Preto jeho závislosti je potrebné určiť ako súčet jeho závislostí spolu so závislosťou knižnice, ktorú využije.

Čím má knižnica menej závislostí, tým je jednoduchšie ju použiť a je stabilnejšia, pretože veľké množstvo knižníc zvyšuje riziko možnej nekompatibility. Z tohto pohľadu je pre matematické výpočty najlepšie použiť knižnicu NumPy, ktorá nevyžaduje žiadne závislosti. Z knižníc strojového učenia je najvýhodnejšie použiť Scikit-learn, ktorá potrebuje

len 4 ďalšie knižnice. V prípade, že kritérium závislosti knižníc je pre používateľa dôležíte, odporúčame sa vyhnúť knižnici TensorFlow, ktorá vyžaduje najviac ďalších knižníc.

4.1.4 Typ licencie

Licenciu open source BSD majú knižnice NumPy, SciPy, PyTorch a Scikit-learn, licenciu open source Apache 2.0 majú knižnice Jax TensorFlow a Keras.

Všetky knižnice sú open source, teda pre užívateľa bezplatne dostupné.

4.2 Výsledky úlohy I. – Riešenie sústavy lineárnych rovníc a inverznej matice

V tejto kapitole porovnáme knižnice na príklade riešenia sústavy lineárnych rovníc a inverznej matice. Porovnanie urobíme na základe času spracovania a presnosti výsledkov.

4.2.1 Sústava lineárnych rovníc

Časové porovnanie

V tabuľke 1 máme zobrazený čas pre každú knižnicu pre výpočty sústavy rovníc rôznych veľkostí, výsledné časy sú zobrazené v milisekundách.

Tab. 1: Čas v ms potrebný pre knižnice na spracovanie rôznych rozmerov matíc.

Rozmer matice	NumPy	SciPy	Jax	PyTorch	TensorFlow
10	0	0	25	16	0
100	66	9	22	3	0
500	94	9	47	3	3
1000	22	22	37	9	47
2500	141	231	122	91	562
5000	900	1291	475	600	3912
7500	3056	4768	1366	1903	12937
10000	7390	11452	3134	4325	31523
15000	22139	34585	9106	14486	103037

Zdroj: Vlastné spracovanie.

Všetky knižnice počítajú inverzné rovnice s rozmermi do 1000x1000 veľmi rýchlo, sú to hodnoty od 0 do 47 milisekúnd čo je menej ako jedna desatina sekundy. Sú to veľmi malé časy, ktoré môžu byt ovplyvnené aj procesmi bežiacimi na pozadí a tak vidíme aj prípady, kde napríklad knižnica PyTorch spočíta sústavu rovníc 500x500 viac ako 5-krát rýchlejšie ako sústavu s rozmerom 10x10.

Významné rozdiely medzi knižnicami pozorujeme pri sústave o veľkosti 5000x5000. Najrýchlejšia knižnica pre tento rozmer je knižnica Jax, za ňou je s malým odstupom PyTorch potom NumPy a SciPy, knižnica TensorFlow začína výrazne zaostávať a výpočty jej trvajú už 3-krát viac ako druhej najpomalšej knižnici. Tento trend pokračuje aj pri všetkých ďalších rozmeroch sústavy rovníc a rozdiely ďalej iba narastajú.

Výpočty inverznej matice väčších rozmerov potvrdí, že najrýchlejšia knižnica je Jax a najpomalšia je TensorFlow.

Porovnanie presnosti

V tabuľke 2 je zobrazená miera nepresnosti jednotlivých knižníc pri riešení sústavy rovníc.

Tab. 2: Miera nepresnosti knižníc pre rôzne rozmery matíc.

Rozmer matice	NumPy	SciPy	Jax	PyTorch	TensorFlow
10	1.332E-15	1.332E-15	4.793E-07	1.243E-15	1.066E-15
100	3.553E-15	3.553E-15	8.696E-07	2.665E-15	2.665E-15
500	6.484E-15	6.484E-15	1.186E-06	4.530E-15	4.263E-15
1000	9.148E-15	9.148E-15	1.373E-06	6.217E-15	5.951E-15
2500	1.537E-14	1.537E-14	1.667E-06	1.021E-14	1.004E-14
5000	1.927E-14	1.927E-14	1.516E-06	1.172E-14	1.128E-14
7500	2.469E-14	2.469E-14	1.693E-06	1.101E-14	1.092E-14
10000	3.046E-14	3.046E-14	1.817E-06	1.137E-14	1.172E-14
15000	3.437E-14	3.437E-14	2.077E-06	1.101E-14	1.288E-14

Zdroj: Vlastné spracovanie.

Nepresnosti výpočtov knižníc trochu stúpajú so zvyšujúcimi sa rozmermi matice ale stále sú veľmi malé a dosahujú hodnoty rádovo 10 na -15 až 10 na -14. Výnimkou je iba knižnica Jax, ktorá ma väčšiu mieru nepresnosti ako ostatné knižnice asi o polovicu.

Napriek horšej presnosti knižnice Jax, je pri riešení väčšiny praktických úloh aj takáto miera akceptovateľná a preto ju nebudeme považovať za výrazný nedostatok knižnice.

Implementačná náročnosť

Rovnice sa počítajú v jednotlivých knižniciach nasledovne:

NumPy:

```
x_numpy = numpy.linalg.solve(A,b)
```

SciPy:

```
x_scipy = scipy.linalg.solve(A,b)
```

Jax:

```
x_jax = jnumpy.linalg.solve(A,b)
```

PyTorch:

```
xTT_torch = torch.linalg.solve(ATT, bTT)
```

TensorFlow:

```
xTF_tf = tf.linalg.solve(ATF, bTF)
```

Všetky knižnice implementujú nájdenie riešenia rovnice jediným riadkom, ktorý je dokonca rovnaký, rozdiely sú len v názve použitej knižnice, z tohto pohľadu je implementačná náročnosť všetkých knižníc je rovnaká.

4.2.2 Výpočet inverznej matice

Časové porovnanie

V nasledujúcej tabuľke 3 máme zobrazené časy pre každú knižnicu pre výpočty inverzných matíc rôznych veľkostí, výsledne časy sú zobrazené v milisekundách.

Tab. 3: Čas v ms potrebný pre rôzne knižnice na spracovanie rôznych rozmerov matíc.

Rozmer matice	NumPy	SciPy	Jax	PyTorch	TensorFlow
10	3	0	19	3	0
100	16	3	25	0	0
500	13	19	41	0	16
1000	44	34	59	31	122
2500	428	369	253	347	1790
5000	2684	2572	1247	2140	13780
7500	8574	8818	3793	7034	49140
10000	19927	21017	8549	15927	120529
15000	63843	68530	27882	51712	417625

Zdroj: Vlastné spracovanie.

Pozorujeme opäť trend, kedy je Jax pri menších rozmeroch najpomalší ale následne sa so zväčšujúcim rozmerom matíc dostáva do popredia a javí sa najrýchlejšou knižnicou. Pri rozmere 15000x15000 je už dokonca 2-krát rýchlejší ako druhá najrýchlejšia knižnica. TensorFlow je najpomalšou knižnicou pri každom rozmere väčšom ako 500x500.Trendy sú veľmi podobné tým pozorovaným pri riešení sústavy rovníc.

Porovnanie presnosti

V nasledujúcej tabuľke 4 je obrazná chybovosť jednotlivých knižníc pri riešení inverzných matíc.

Tab. 4: Miera nepresnosti knižníc pre rôzne rozmery matíc.

Rozmer matice	NumPy	SciPy	Jax	PyTorch	TensorFlow
10	3.109E-16	2.887E-16	1.597E-07	3.331E-16	3.775E-16
100	1.088E-15	1.132E-15	2.096E-07	1.177E-15	1.310E-15
500	1.932E-15	1.954E-15	3.204E-07	1.932E-15	1.843E-15
1000	1.799E-15	1.799E-15	3.400E-07	1.932E-15	1.843E-15
2500	2.509E-15	2.420E-15	4.358E-07	2.509E-15	2.398E-15
5000	2.709E-15	2.576E-15	2.587E-07	2.420E-15	2.842E-15
7500	3.064E-15	3.375E-15	1.319E-07	2.998E-15	3.153E-15
10000	3.442E-15	3.264E-15	1.054E-07	3.064E-15	3.686E-15
15000	3.864E-15	3.597E-15	5.419E-08	3.753E-15	4.219E-15

Zdroj: Vlastné spracovanie.

Nepresnosti výpočtov trochu stúpajú so zvyšujúcimi sa rozmermi matice ale stále sú veľmi malé a dosahujú hodnoty rádovo 10 na -16 až 10 na -15. Jax podobne ako pri sústave rovníc má o polovicu hrošie výsledky ako ostatné knižnice. Takéto rozdiely považujeme za zanedbateľné a knižnice majú prakticky rovnakú presnosť výpočtu.

Implementačná náročnosť

Inverzná matica sa počíta v jednotlivých knižniciach nasledovne:

NumPy:

```
Ainv_numpy = numpy.linalg.inv(A)
```

SciPy:

```
Ainv_scipy = scipy.linalg.inv(A)
```

Jax:

```
Ainv_jax = jnumpy.linalg.inv(A)
```

PyTorch:

```
ATTinv_torch = torch.linalg.inv(ATT)
```

TensorFlow:

```
ATTinv_tf = tf.linalg.inv(ATF)
```

Všetky knižnice implementujú nájdenie inverznej matice jediným riadkom, ktorý je dokonca rovnaký, rozdiely sú len v názve použitej knižnice, z tohto pohľadu implementačná náročnosť všetkých knižníc je rovnaká.

Pre knižnice PyTorch a TensorFlow sa vykonala aj transformácia dát do tensorov čo však nie je nevýhodou týchto knižníc, preto že ak by vstupné dáta boli vytvárané vo forme tensorov, tak by naopak knižnice NumPy a SciPy vyžadovali transformáciu týchto dát do ich štruktúr.

4.2.3 Celkové zhodnotenie

Pri riešení sústav lineárnych rovníc a inverznej matice pre malé rozmery matice (do 1000x1000) si môžeme vybrať ktorúkoľvek z porovnávaných knižníc, nakoľko sú ich časy výpočtu tak malé že rozdiely môžeme zanedbať. Okrem knižnice Jax, ktorá sa javí najnevýhodnejšou pre svoju horšiu mieru presnosti.

V prípade väčších rozmerov matíc by sme pre ušetrenie času je najvýhodnejšie použiť knižnicu Jax pretože je najrýchlejšia. To však platí len pre prípad, kde jej väčšia miera nepresnosti nezaváži. Vyhnúť by sme sa mali knižnici TensorFlow, nakoľko je výrazne pomalšia oproti ostatným.

Zaujímavá je knižnica NumPy, ktorá síce nie je najrýchlejšia ale má vysokú presnosť a nemá žiadne závislosti od ostatných knižníc.

4.3 Výsledky úlohy II. – Riešenie klasifikácie kvetov pomocou neurónovej siete

4.3.1 Porovnanie času učenia

V tabuľke 5 je uvedený čas v milisekundách, ktorý bol potrebný pre učenie jednotlivých knižníc.

Tab. 5: Čas učenia v ms pri klasifikácii datasetu Iris.

Epochy	Scikit-learn	PyTorch	TensorFlow	Keras +Jax	Keras +PyTorch	Keras +TensorFlow
10	19	28	1075	731	228	1016
50	31	66	2003	822	635	1956
100	66	109	3319	981	1253	3084
500	81	644	13518	2253	6345	12584
1000	122	1191	26308	3862	12205	24595
2500	91	3315	65633	8662	30473	60068
5000	78	5909	126476	16680	61343	118498
7500	87	8877	191702	24805	92122	178369
10000	106	11340	255285	32735	122782	236121

Zdroj: Vlastné spracovanie.

Môžeme pozorovať že už pri najmenšom počte epoch je čas potrebný pre učenie modelu rozdielny pre porovnávané knižnice. Kým Scikit-learn potrebuje na 10 epoch len 19 msc, najpomalšia knižnica TensorFlow potrebuje viac ako sekundu. So zväčšujúcim sa počtom epoch čas učenia pre všetky knižnice rastie lineárnymi spôsobom, výnimkou je iba knižnica Scikit-learn, pre ktorú je čas učenia veľmi malý a dosahuje jednu desatinu sekundy. Toto je spôsobené tým, že pri tejto knižnici parameter počet epoch neznamená počet epoch, ktoré knižnica použije ale znamená maximálny počet epoch, ktoré môže knižnica využiť. Skutočný počet epoch si však knižnica určuje sama podľa jej vlastných kritérií.

V tabuľke 5 môžeme pozorovať aj aký vplyv má Keras na iné knižnice. Knižnica TensorFlow je najpomalšia keď sa používa samostatne, ale v kombinácii s Kerasom je o málo rýchlejšia. Naopak, knižnica PyTorch je veľmi rýchla avšak spolu s Kerasom je až 10 krát pomalšia.

4.3.2 Porovnanie presnosti klasifikácie

V tabuľke 6 môžeme vidieť akú presnosť dosahujú jednotlivé knižnice pri vyhodnocovaní testovacej a trénovacej časti datasetu.

Tab. 6: Miera nepresnosti pri klasifikácii datasetu Iris.

	Scikit-learn		PyTorch		TensorFlow		Keras+Jax		Keras+PyTorch		Keras+TensorFlow	
Epochy	test	train	test	train	test	train	test	train	test	train	test	train
10	0.768	0.746	0.705	0.705	0.926	0.914	0.963	0.952	0.958	0.955	0.974	0.970
50	0.989	0.977	0.995	0.973	0.989	0.971	0.974	0.971	0.979	0.979	0.979	0.980
100	1.000	0.971	1.000	0.979	0.995	0.975	0.974	0.982	0.968	0.971	0.984	0.982
500	1.000	0.982	1.000	0.980	0.995	0.971	0.974	0.988	0.974	0.986	0.979	0.980
1000	1.000	0.980	1.000	0.977	1.000	0.975	0.974	0.991	0.958	0.984	0.963	0.984
2500	1.000	0.975	1.000	0.986	0.995	0.963	0.974	0.995	0.968	0.982	0.958	0.984
5000	1.000	0.975	1.000	0.984	1.000	0.977	0.974	0.993	0.968	0.982	0.937	0.988
7500	1.000	0.982	1.000	0.982	1.000	0.982	0.974	0.993	0.958	0.984	0.932	0.988
10000	1.000	0.982	1.000	0.989	1.000	0.982	0.916	0.929	0.953	0.982	0.953	0.989

Zdroj: Vlastné spracovanie.

Všetky knižnice dosahujú výbornú presnosť klasifikácie kvetov v datasete Iris s výnimkou knižníc Scikit-learn a PyTorch pre najmenší počet epoch 10. Aj tieto knižnice však s väčším počtom epoch presnosť zlepšujú a pre testovanie dáta dokonca majú 100 percentu úspešnosť klasifikácie spolu s knižnicou TensorFlow. Niektoré knižnice dosahujú väčšiu presnosť na testovacích a niektoré na trénovacích dátach, tie ktoré dosahujú väčšiu presnosť na testovacích sú vždy knižnice ktoré využívajú Keras. Zaujímavá situácia nastáva pri počte epoch 10000, kde Jax zaznamenáva pokles v presnosti oproti predchádzajúcim počtom epoch.

4.3.3 Porovnanie času klasifikácie objektov

V tabuľke 7 sú údaje o čase, ktorý potrebujú knižnice pre klasifikáciu testovacích a trénovacích dát.

Tab. 7: Čas klasifikácie v ms pre dataset Iris.

	Scikit-learn		PyTorch		TensorFlow		Keras+Jax		Keras+PyTorch		Keras+TensorFlow	
Epochy	test	train	test	train	test	train	test	train	test	train	test	train
10	0	0	0	0	50	122	37	91	2	4	44	97
50	0	0	0	0	47	94	37	78	2	4	44	94
100	0	0	3	0	44	94	41	81	2	4	44	94
500	0	0	0	0	47	94	37	87	1	2	44	94
1000	0	0	0	0	50	94	34	84	0	3	44	91
2500	0	3	0	0	47	97	44	81	3	6	47	87
5000	0	0	0	0	47	91	41	84	3	3	47	91
7500	0	0	0	0	47	94	41	78	3	3	44	91
10000	0	0	0	0	47	94	41	78	3	0	47	88

Zdroj: Vlastné spracovanie.

Pozorujeme, že všetky knižnice dokážu klasifikovať dáta veľmi rýchlo, Scikit-learn a PyTorch dokonca za čas menší ako milisekundu.

Čas klasifikácie je pre praktické použitie knižnice dôležitejší ako čas učenia modelu pretože v skutočných aplikáciách sa môže model učiť aj niekoľko týždňov, dôležité však je, aby dokázal rýchlo a správne klasifikovať neznáme objekty. Z tohto pohľadu sú všetky knižnice vhodné na použitie pri praktických úlohách strojového učenia.

4.3.4 Porovnanie obtiažnosti implementácie

Využitie neurónovej siete v algoritme pozostáva z niekoľkých krokov. Najprv sa zadefinuje model neurónovej siete, ktorý sa potom skompiluje, prípadne sa mu nastavia ďalšie parametre. Keď je model pripravený, prebehne fáza učenia sa a nakoniec sa model použije na klasifikáciu dát.

Scikit-learn:

```
# model with 2 hidden layers 25 + 30 nodes
model = MLPClassifier(solver='adam', hidden_layer_sizes = [25,30],
        learning_rate_init=0.01, max_iter=epoch)

# learn model
model.fit(X_train, y_train)

# use model for X_test data
y_predict_test = model.predict(X_test)
```

PyTorch:

```
# model with 2 hidden layers 25 + 30 nodes
class Model(torch.nn.Module):
    def __init__(self, input_features=4, hidden_layer1=25,
                 hidden_layer2=25, output_features=3):
        super().__init__()
        self.fc1 = torch.nn.Linear(input_features,hidden_layer1)
        self.fc2 = torch.nn.Linear(hidden_layer1, hidden_layer2)
        self.out = torch.nn.Linear(hidden_layer2, output_features)

    def forward(self, x):
        x = torch.nn.functional.relu(self.fc1(x))
        x = torch.nn.functional.relu(self.fc2(x))
        x = self.out(x)
        return x
```

```
model = Model()
criterion = torch.nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.01)

# learn model
losses = []
for i in range(epoch):
    y_pred = model.forward(Xt_train)
    loss = criterion(y_pred, yt_train)
    losses.append(loss)
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

# use model for X_test data
with torch.no_grad():
    yt_predict_test = model(Xt_test)
```

TensorFlow:

```
# model with 2 hidden layers 25 + 30 nodes
model = tf.keras.Sequential()
model.add(tf.keras.Input(shape=(4,)))
model.add(tf.keras.layers.Dense(25, activation='relu'))
model.add(tf.keras.layers.Dense(30, activation='relu'))
model.add(tf.keras.layers.Dense(3, activation='softmax'))
model.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=0.01),
    loss=tf.keras.losses.BinaryCrossentropy(),
    metrics=[tf.keras.metrics.Accuracy()],
)

# learn model
model.fit(Xtf_train, ytf_train, epochs=epoch)

# use model to predict test
ytf_predict_test = model.predict(Xtf_test)
```

Keras (pre všetky knižnice ktoré používa):

```
# model with 2 hidden layers 25 + 30 nodes
model = keras.models.Sequential()
model.add(keras.Input(shape=(4,)))
model.add(keras.layers.Dense(25, activation='relu'))
model.add(keras.layers.Dense(30, activation='relu'))
model.add(keras.layers.Dense(3, activation='softmax'))

model.compile(
    optimizer=keras.optimizers.Adam(learning_rate=0.01),
    loss=keras.losses.BinaryCrossentropy(),
    metrics=['accuracy'],
)

# learn model
model.fit(X_train, train_labels, epochs=epoch, verbose=0)

# use model to predict test
yy_predict_test = model.predict(X_test, verbose=0)
```

Model neurónovej siete sa najjednoduchšie vytvorí a pripraví na fázu učenia pomocou knižnice Scikit-learn, kde stačí jeden riadok kódu. Naopak, najťažšie použiteľná knižnica je PyTorch, ktorá na vytvorenie a prípravu modelu potrebuje až 14 riadkov kódu. Knižnica TensorFlow a Keras (so všetkými potrebnými knižnicami) sú z pohľadu obtiažnosti implementácie rovnaké, dôvod je, že knižnica TensorFlow sama ma závislosť na knižnici Keras a využíva jeho príkazy aj pre svoju priamu implementáciu.

Pre fázu učenia, knižniciam Scikit-learn, TensorFlow a Keras stačí jeden riadok pričom knižnice TensorFlow a Keras tu nastavujú parametre učenia. V knižnici Scikit-learn sa tieto parametre nastavujú pri definícii modelu. Knižnica PyTorch potrebuje až 8 príkazov.

Fáza použitia modelu pre klasifikáciu objektov je v knižniciach Scikit-learm TensorFlow a Keras implementovaná jedným riadkom, knižnica PyTorch potrebuje 2 riadky kódu.

Za najjednoduchšie použiteľnú knižnicu teda považujeme Scikit-learn, za ňou nasledujú knižnice TensorFlow, Keras + Jax, Keras + PyTorch, Keras + TensorFlow , a ako najhoršiu ohodnocujeme na základe obtiažnosti implementácie knižnicu PyTorch.

4.3.5 Celkové zhodnotenie

Pri klasifikácii dát z datasetu Iris je najvhodnejšie používať knižnicu Scikit-learn pretože je najrýchlejšia, dosahuje výbornú presnosť klasifikácie (ak sa počet epoch nastaví na viac ako 50) a je najjednoduchšie implementovateľná.

Vyhnúť by sme sa mali knižnici Tensor flow aj jej použitiu s knižnicou Keras, nakoľko je najpomalšia. V prípade, že by bolo potrebné použiť vysoký počet epoch, je vhodná aj knižnica PyTorch, ktorá je najrýchlejšia, aj keď ma najvyššiu obtiažnosť použitia.

4.4 Výsledky úlohy III. – Riešenie klasifikácie rukou písaných číslic pomocou neurónovej siete

4.4.1 Porovnanie času učenia

V tabuľke 8 je uvedený čas v milisekundách, ktorý bol potrebný pre učenie jednotlivých knižníc.

Tab. 8: Čas učenia v ms pri klasifikácii datasetu Mnist.

Epochy	Scikit-learn	PyTorch	TensorFlow	Keras +Jax	Keras +PyTorch	Keras +TensorFlow
10	23751	897	12421	26693	58565	12215
50	116995	4640	61393	133226	290155	60265
100	124273	9521	122982	268965	577088	118323
200	125676	17499	251489	504996	1153926	239062
300	131763	25579	386976	764660	1723896	354066
400	123720	33695	530591	1067582	2315946	473949
500	127288	44144	676731	1287780	2877510	592650

Zdroj: Vlastné spracovanie.

Môžeme pozorovať už pri najmenšom počte epoch, že čas potrebný pre učenie modelu je pre porovnávané knižnice rozdielny. Kým PyTorch potrebuje na 10 epoch 897 msc, najpomalšia knižnica PyTorch S Kerasom potrebuje viac ako 58 sekúnd čo je 60-krát viac.

Podobne ako pri riešení úlohy II s datasetom IRIS aj tu s rastúcim počtom epoch čas učenia pre všetky knižnice, okrem knižnice Scikit-learn, rastie lineárnym spôsobom. Aj v tomto prípade knižnica Keras ovplyvňuje výsledky iných knižníc. Keď sa knižnica Tensor Flow používa samostatne je najpomalšia knižnica a v kombinácii s Kerasom je o málo rýchlejšia. Naopak, knižnica PyTorch je osamote veľmi rýchla, avšak spolu s Kerasom je až 65-krát pomalšia.

4.4.2 Porovnanie presnosti klasifikácie

V tabuľke 9 sú uvedené výsledky z riešenia klasifikačnej úlohy s riešením datasetu Mnist pomocou neurónovej siete. Podobne ako s datasetom Iris, aj tu je v jednotlivých stĺpcoch uvedená presnosť klasifikácie zistená na testovacej a trénovanej skupine dát ako aj čas potrebný na učenie modelu pre každú z porovnávaných knižníc.

Tab. 9: Miera nepresnosti pri klasifikácii datasetu Mnist.

	Scikit-learn		PyTorch		TensorFlow		Keras+Jax		Keras+PyTorch		Keras+TensorFlow	
Epochy	**test**	**train**	**test**	**train**	**test**	**train**	**test**	**train**	**test**	**train**	**test**	**train**
10	0.973	0.988	0.473	0.463	0.949	0.949	0.956	0.960	0.955	0.959	0.948	0.949
50	0.974	1.000	0.802	0.795	0.971	0.981	0.974	0.989	0.974	0.990	0.971	0.981
100	0.975	1.000	0.863	0.856	0.975	0.989	0.974	0.997	0.974	0.997	0.974	0.989
200	0.974	1.000	0.896	0.889	0.975	0.996	0.975	1.000	0.973	1.000	0.975	0.996
300	0.975	1.000	0.902	0.898	0.975	0.999	0.974	1.000	0.974	1.000	0.975	0.998
400	0.975	1.000	0.909	0.905	0.974	0.999	0.973	1.000	0.974	1.000	0.975	0.999
500	0.975	1.000	0.915	0.911	0.975	1.000	0.975	1.000	0.974	1.000	0.975	1.000

Zdroj: Vlastné spracovanie.

Všetky knižnice dosahujú výbornú presnosť klasifikácie číslic Mnistu s výnimkou knižnice PyTorch pre najmenší počet epoch 10. Správne vyhodnotila menej ako polovicu zadaných dát, so zvýšením počtu epoch presnosť zvyšuje takmer dvojnásobne na 80%-nú úspešnosť, avšak pri porovnaní s ostatnými knižnicami sú výsledky nedostačujúce, preto ju pre tento dataset použiť neodporúčame.

Pri datasete Mnist všetky knižnice dosahujú lepšiu presnosť na trénovacích dátach okrem PyTorchu. Najlepšie výsledky na nich má knižnica Scikit-learn, ktorá už na päťdesiatich epochách dosahuje 100% úspešnosť vyhodnocovania výsledkov.

Pozorujeme lepšie fungovanie knižnice PyTorch, keď je použitá pomocou knižnice Keras, jej presnosť je teraz na rovnakej úrovni s ostatnými, TensorFlow knižnicou Keras nie je až tak ovplyvnená a hovoríme o skoro rovnakých výsledkoch.

4.4.3 Porovnanie času klasifikácie objektov

V tabuľke 10 je čas v milisekundách, ktorý potrebuje príslušná knižnica na klasifikáciu dát pomocou modelu neurónovej siete, ktorý už proces učenia vykonal. Je tu uvedený čas pre klasifikáciu všetkých dát z testovacej časti dát ako aj z trénovanej časti.

Tab. 10: Čas klasifikácie v ms pre dataset Mnist.

	Scikit-learn		PyTorch		TensorFlow		Keras+Jax		Keras+PyTorch		Keras+TensorFlow	
Epochy	test	train	test	train	test	train	test	train	test	train	test	train
10	31	219	6	34	416	1747	228	916	306	1716	381	1791
50	37	216	9	28	362	1712	222	926	281	1662	381	1778
100	31	219	9	31	441	1734	233	968	278	1662	375	1794
200	31	219	6	34	366	1803	212	891	269	1609	372	1797
300	34	216	0	31	362	1778	228	928	281	1616	375	1812
400	34	216	0	34	369	1769	237	1012	278	1734	375	1797
500	28	212	6	31	372	1778	231	925	300	1650	375	1850

Zdroj: Vlastné spracovanie.

Čas klasifikácie je pre všetky knižnice približne rovnaký pre všetky hodnoty epoch, čo je celkom prirodzené, pretože počet epoch ovplyvňuje len proces učenia ale nie následné použitie modelu pre klasifikáciou objektov.

Pri každej knižnici je čas potrebný na klasifikáciu testovacej časti dát menší ako čas potrebný na klasifikáciu trénovacej časti dát. Tento rozdiel je spôsobený tým, že testovacích dát je 10 000 a trénovaniach 60 000 a pomer 1:6 je viditeľný aj pri porovnaní časov klasifikácie.

4.4.4 Porovnanie obtiažnosti implementácie

Podobne ako pri datasete Iris, aj tu porovnáme riadky zdrojového kódu, ktorý je potrebný pre vytvorenie modelu, jeho učenie a použitie pre klasifikáciu objektov.

Scikit-learn:

```
# model with 1 hidden layers 50 nodes
model = MLPClassifier(solver='sgd', hidden_layer_sizes = [50],
                      learning_rate_init=0.1, max_iter=epoch)

# learn model
model.fit(X_train, y_train)

# use model for X_test data
y_predict_test = model.predict(X_test)
```

PyTorch:

```
# model with 1 hidden layers 50 nodes
class Model(torch.nn.Module):
```

```
        def __init__(self, input_features=784, hidden_layer1=50,
                     output_features=10):
            super().__init__()
            self.fc1 = torch.nn.Linear(input_features,hidden_layer1)
            self.out = torch.nn.Linear(hidden_layer1, output_features)

        def forward(self, x):
            x = torch.nn.functional.relu(self.fc1(x))
            x = self.out(x)
            return x

    model = Model()
    criterion = torch.nn.CrossEntropyLoss()
    optimizer = torch.optim.SGD(model.parameters(), lr=0.1)

    # learn model
    losses = []
    for i in range(epoch):
        y_pred = model.forward(Xt_train)
        loss = criterion(y_pred, yt_train)
        losses.append(loss)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

    # use model for X_test data
    with torch.no_grad():
        yt_predict_test = model(Xt_test)
```

TensorFlow:

```
    # model with 1 hidden layers 50 nodes
    model = tf.keras.Sequential()
    model.add(tf.keras.Input(shape=(784,)))
    model.add(tf.keras.layers.Dense(50, activation='relu'))
    model.add(tf.keras.layers.Dense(10, activation='softmax'))
    model.compile(
        optimizer=tf.keras.optimizers.SGD(learning_rate=0.1),
        loss=tf.keras.losses.BinaryCrossentropy(),
        metrics=[tf.keras.metrics.Accuracy()],
```

```
    )

    # learn model
    model.fit(Xtf_train, ytf_train, epochs=epoch)

    # use model to predict test
    ytf_predict_test = model.predict(Xtf_test)
```

Keras (pre všetky knižnice ktoré používa):

```
    # model with 1 hidden layers 50 nodes
    model = keras.models.Sequential()
    model.add(keras.Input(shape=(784,)))
    model.add(keras.layers.Dense(50, activation='relu'))
    model.add(keras.layers.Dense(10, activation='softmax'))

    model.compile(
        optimizer=keras.optimizers.SGD(learning_rate=0.1),
        loss=keras.losses.BinaryCrossentropy(),
        metrics=['accuracy'],
    )

    # learn model
    model.fit(X_train, train_labels, epochs=epoch, verbose=0)

    # use model to predict test
    yy_predict_test = model.predict(X_test, verbose=0)
```

Príkazy pre klasifikáciu objektov datasetu Mnist v jednotlivých knižniciach sú veľmi podobné príkazom pre klasifikáciu Iris. Je to spôsobené tým, že knižnice v obidvoch prípadoch riešia klasifikačnú úlohu, to znamená že príkazy na učenie ako aj klasifikáciu dát sú úplne rovnaké. Odlišné sú len príkazy na vytvorenie príslušného modelu, nakoľko model neurónovej siete pre dataset Mnist mal len 1 skrytú vrstvu s väčším počtom uzlov v porovnaní s modelom pre dataset Iris, kde boli skryté vrstvy dve.

Za najjednoduchšiu na implementáciu považujeme knižnicu SciKti-Learn, následne sú knižnice TensorFlow, Keras + Jax, Keras + PyTorch, Keras + TensorFlow, ktoré majú takmer rovnakú náročnosť použitia, najťažšia na implementáciu bola aj v tomto prípade knižnica PyTorch.

4.4.5 Celkové zhodnotenie

Na klasifikáciu číslic z datasetu Mnist nie je vhodné použiť knižnicu PyTorch, pretože napriek jej vysokej rýchlosti aj na učenie aj na klasifikáciu dát jej presnosť je v porovnanie s ostatnými nedostačujúca, hlavne pri menších počtoch epoch. Naopak Scikit-learn prejavuje vysokú úspešnosť klasifikácie a jej rýchlosť je lepšia v porovnaní s ostatnými knižnicami. Tie vyžadujú oveľa viac času na učenie ako aj na následnú klasifikáciu, preto by sa mohli použiť v prípade, že čas nie je až tak podstatný.

Zároveň by sme ale čitateľov radi upozornili a pripomenuli im fakt, že sme na testovanie použili dataset, ktorý je predurčený na učenie a tým pádom sa dá predpokladať, že bude algoritmus dosahovať lepšie výsledky ako by boli dosiahnuté na vlastnom projekte. Pri vytváraní vlastného datasetu teda môžeme očakávať nižšiu presnosť u všetkých knižníc, takémuto porovnaniu sme sa ale nevenovali, nakoľko to nebolo cieľom našej práce.

5 DISKUSIA

V diskusii by sme čitateľom radi pripomenuli, že sme nevyčerpali všetky možnosti porovnania knižníc a na problematiku sme nahliadli iba čiastočne.

Rozšírením tejto práce by mohlo byť testovanie a porovnávanie knižníc na ďalších datasetoch, ktoré by mohli byť ešte rozsiahlejšie ako Mnist. Takéto datasety by zodpovedali typu úloh, ktoré sú aktuálne riešené v praxi. Ďalej si predstavíme ďalšie možne smery rozšírenia práce alebo podobných prác.

5.1 Vplyv parametrov knižníc na ich výkon

V praktickej časti našej prace sme sa snažili dosiahnuť rôzne výsledky pomocou zmeny počtu epoch, to ale nie je jediný nastaviteľný parameter, ktorý knižnice ovplyvňuje, Mohlo sa stať, že na štandardizovaných parametroch jednoducho fungujú niektoré knižnice lepšie a niektoré podstatne horšie. Bolo by potrebné veľa testovať ďalšími pokusmi aké nastavenia sú vhodné pre ktorú knižnicu a následne ich porovnávať s najoptimálnejším nastavením.

5.2 Test presnosti knižníc na dátach ktoré nie sú súčasťou datasetu

V našej práci sme overovali presnosť len na dátach, ktoré boli súčasťou datasetu. Knižnice by sme mohli porovnávať aj na klasifikácii nami vytvorených dát, ktoré by sa mohli vytvárať rôznym spôsobom tak aby boli viac alebo menej podobne dátam v datasete.

5.3 Porovnanie knižníc na rôznych algoritmoch strojového učenia

V našej práci sme sa rozhodli používať iba knižnice, ktoré na strojové učenie používajú neurónové siete, avšak v práci spomíname existenciu aj iných algoritmov použiteľných pre strojové učenie. Knižnice by sa dali porovnávať aj na takýchto ďalších algoritmoch.

5.4 Porovnanie knižníc v ďalších oblastiach strojového učenia

V našej práci sme sa zamerali len na porovnanie knižníc z pohľadu strojového učenia a riešenia matematických úloh, ktoré sa pri strojovom učení využívajú. Problematika strojového učenia je však širšia a zahŕňa aj oblasti prípravy dát, ich normalizácie, štatistickej analýzy, zobrazovania a ďalšie. Tieto oblasti sú tiež podporované rôznymi knižnicami, ktoré by sa mohli porovnať.

5.5. Porovnanie knižníc na ďalších úlohách strojového učenia

V našej práci sme sa zamerali len na klasifikačnú úlohu strojového učenia s učiteľom. Knižnice by sa mohli porovnávať aj na úlohách strojového učenia bez učiteľa alebo na úlohe reinforcement. Pri strojovom učení s učiteľom by sa dali porovnať aj pre riešenie regresnej úlohy.

ZÁVER

Cieľom tejto práce bolo porovnať rôzne knižnice v jazyku Python slúžiace na strojové učenie, čitateľa oboznámiť s ich rozdielmi a priblížiť mu problematiku strojového učenia. Tieto stanovené ciele sme detailne rozpracovali v rôznych častiach tejto práce.

V teoretickej časti sme objasnili problematiku ako aj témy blízke strojovému učeniu, identifikovali knižnice a popísali niektoré ich vlastnosti. Tu sme knižnice porovnali na základe informácii získaných z ich domovských stránok.

V praktickej časti sme porovnali knižnice na riešení konkrétnych klasifikačných úlohách strojového učenia s učiteľom. Tu sme našli konkrétne výhody a nevýhody jednotlivých knižníc, ktoré sa prejavili počas ich použitia, následne sme z výsledkov vyvodili závery a knižnice sme porovnali. Niektoré výsledky boli prekvapujúce, iné naplnili naše očakávania. Prekvapujúce bolo, že knižnice sú veľmi výkonné, napríklad všetky trénovanie dáta v datasete MNIST, ktorých bolo 60 000 sa im podarilo klasifikovať v čase niekoľkých desiatok milisekúnd až dvoch sekúnd. Tiež prekvapujúca bola presnosť klasifikácie, ktorá dosahovala úroveň viac ako 91% a to aj pri najhorších knižniciach, pričom najlepšie dosahovali aj úroveň bez jedinej chyby.

Výsledky riešenia matematických úloh poukazujú aj na efektivitu využitia knižníc a technológií na ich riešene. Vlastnoručné riešenie rovníc je náročne už pri rozmeroch 3x3, ale s pomocou knižníc aj sústavy s 15 000 rovnicami sa spočítajú v čase 9 sekúnd pričom presnosť výpočtu je 14-15 desatinných miest. Keď zoberieme do úvahy rôzne kritéria, môžeme vyzdvihnúť knižnicu Scikit-learn. Táto knižnica sa najjednoduchšie používa, dosiahla vysokú presnosť klasifikácie tak pri riešení datasetu Iris ako aj Mnist, pričom hlavne pri malom datasete Iris dosiahla najlepší čas. Knižnica Keras umožňuje vytvoriť jeden program, ktorý sa jednoduchým spôsobom dá použiť s rôznymi ďalšími knižnicami, ktoré môžu byt vhodnejšie na rôzne úlohy. To nám dovoľuje vytvoriť jeden program, ktorý je následne prispôsobiteľný na projekty iného typu bez väčších úprav. Takto sa potvrdilo naše očakávanie, že žiadnu knižnicu nemôžeme označiť za najlepšiu vo všetkých oblastiach. Na rôzne úlohy sú vhodné rôzne knižnice, pričom výber silne závisí aj od toho, kto knižnice bude používať za akým účelom. Niektoré knižnice sú vhodné pre začiatočníkov pre jednoduchosť implementácie, iné sú skôr vhodné na riešenie úloh, kde je nevyhnutná vysoká presnosť a ďalšie tam, kde je najdôležitejší čas na klasifikáciu prípadne čas na učenie.

ZOZNAM POUŽITEJ LITERATÚRY

[1] BROWN, Sara. Machine learning, explained [Online]. 2021. Dostupné na internete: https://mitsloan.mit.edu/ideas-made-to-matter/machine-learning-explained

[2] GAUTAM, Anjali, et al. Deep learning approach to automatically recognise license number plates. Multimedia Tools and Applications, 2023, 82.20: 31487-31504.

[3] SHINDE, Pramila P.; SHAH, Seema. A review of machine learning and deep learning applications. In: 2018 Fourth international conference on computing communication control and automation (ICCUBEA). IEEE, 2018. p. 1-6.

[4] OSHODI, Ismaila, Kolawole. Machine Learning-based Algorithms for Weather Forecasting. [Online]. 2022. Dostupné na internete:

https://www.researchgate.net/publication/361721049_Machine_learning-based_algorithms_for_weather_forecasting

[5] MURÁŇ, Juraj. Algoritmy strojového učenia I. – Učenie s učiteľom. [Online]. 2019. Dostupné na internete: https://umelainteligencia.sk/algoritmy-strojoveho-ucenia/

[6] CHINA, Chrystal R. Five machine learning types to know. [Online]. 2024. Dostupné na internete: https://www.ibm.com/blog/machine-learning-types/

[7] AKSENOV, Sergey. What is supervised learning? Machine learning tasks. [Online]. 2023. Dostupné na internete: https://www.superannotate.com/blog/supervised-learning-and-other-machine-learning-tasks

[8] Müller, Andreas C.; GUIDO, Sarah. Introduction to Machine Learning with Python. 4. vyd.Sebastopol: O‘Reilly Media, 2018. 384 s. ISBN:978-1-449-36941-5

[9] I. Neurónové siete – základy. [Online]. Dostupné na internete: https://course.elementsofai.com/sk/5/1

[10] MURÁŇ, Juraj. Úvod do neurónových sietí. [Online]. 2019. Dostupné na internete: https://umelainteligencia.sk/uvod-do-neuronovych-sieti/

[11] Epoch in Neural Networks. [Online]. 2024. Dostupné na internete: https://www.baeldung.com/cs/epoch-neural-networks

[12] HILLIER, Will.What's the Best Language for Machine Learning? [Online]. 2023. Dostupné na internete: https://careerfoundry.com/en/blog/data-analytics/best-machine-learning-languages/

[13] DAIGLE, Kyle. The state of open source and rise of AI in 2023: Top 10 programming languages on GitHub. [Online]. 2023. Dostupné na internete: https://github.blog/2023-11-08-the-state-of-open-source-and-ai/#the-most-popular-programming-languages

[14] OSTROWSKA, Kamila. A Brief History of Python [Online]. 2022. Dostupné na internete: https://learnpython.com/blog/history-of-python/

[15] STANČIN, Igor; JOVIĆ, Alan. An overview and comparison of free Python libraries for data mining and big data analysis. In: 2019 42nd International convention on information and communication technology, electronics and microelectronics (MIPRO). IEEE, 2019. p. 977-982.

[16] BOZKUS Emine. Exploring the Iris flower dataset. [Online]. 2023. Dostupné na internete: https://eminebozkus.medium.com/exploring-the-iris-flower-dataset-4e000bcc266c

[17] OpenML: iris. [Online]. Dostupné na internete:

https://www.openml.org/search?type=data&status=active&id=61&sort=runs

[18] ERSOZ, Ozgur. Iris Flowers Classification with PyTorch. [Online]. 2020. Dostupné na internete: https://medium.com/@ozgur.ersoz3/iris-flowers-classification-with-pytorch-cd80c8aeeb2c

[19] OpenML: mnist_784. [Online]. Dostupné na internete:

https://www.openml.org/search?type=data&status=active&id=554&sort=runs

[20] LECUN, Yann; CORTES, Corinna; BURGES, Christopher J.C. The Mnist database of handwritten digit. [Online]. Dostupné na internete: http://yann.lecun.com/exdb/mnist/

[21] DECOSTE, Dennis; CRISTIANINI, Nello. Training Invariant Support Vector Machines. [Online]. 2003. Dostupné na internete: https://www.researchgate.net/figure/The-first-100-MNIST-training-images-with-class-labels_fig1_2924845

[22] LU, Johnny(ZHIPING). Using Scikit-Learn Neural Network Class to classify MNIST. [Online]. 2020. Dostupné na internete: https://johdev.com/jupyter/2020/03/02/Sklearn_MLP_for_MNIST.html