JOURNAL
COMPUTER SCIENCES INSTITUTE

JCSI 33 (2024) 339–345
Received: 17 September 2024
Accepted: 8 October 2024

Performance of machine learning tools. Comparative analysis of libraries in interpreted and compiled programming languages

Wydajność narzędzi uczenia maszynowego. Analiza porównawcza bibliotek w interpretowanych i kompilowanych językach programowania

Tomasz Wiejak*, Jakub Smołka

Department of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland

Abstract

The article compares machine learning tools using the example of several popular programming languages. Existing tools in the following programming languages were tested and compared with each other: Python, Java, R, Julia, C#. For the needs of article, algorithms were created in each studied language, operating on the same test set and using algorithms from the same group. The collected results included the program's running time, number of lines of code and accuracy of trained model. Based on the obtained data, conclusions were drawn that interpreted language libraries in terms of creating machine learning solutions are more effective than compiled language libraries.
Keywords: machine learning; interpreted language; compiled language

Streszczenie

W artykule porównano narzędzia uczenia maszynowego na przykładzie kilku popularnych języków programowania. Przetestowano i porównano istniejące narzędzia w następujących językach programowania: Python, Java, R, Julia, C#. Na potrzeby artykułu w każdym badanym języku zostały stworzone algorytmy operujące na tym samym zbiorze testowym i wykorzystujące algorytmy z tej samej grupy. Rejestrowane wartości obejmowały czas działania programu, liczbę linii kodu, jak i dokładność uczenia modeli. Na podstawie wszystkich otrzymanych danych wyciągnięto wnioski, a w rezultacie stwierdzono, że biblioteki języków interpretowanych pod względem tworzenia rozwiązań uczenia maszynowego są skuteczniejsze niż biblioteki języków kompilowanych.
Słowa kluczowe: uczenie maszynowe; język interpretowany; język kompilowany

*Corresponding author
Email address: **tomasz.wiejak@pollub.edu.pl** (T. Wiejak)

1. Wstęp

Obecnie informatyka jest jedną z bardziej cenionych dziedzin. Codziennie można dowiedzieć się o nowych dokonaniach naukowców, jak i programistów. Powstają coraz to bardziej zaawansowane narzędzia operujące na tekście, muzyce, obrazach, a także i na filmach. Aby to wszystko było możliwe, wymagane są dokładne i wydajne narzędzia uczenia maszynowego, które przetwarzają jak największą ilość danych w jak najkrótszym czasie. Istnieje wiele bibliotek i dodatków do każdego języka umożliwiające pracę ze zbiorami danych pod kątem uczenia maszynowego. Niewątpliwie najpopularniejszym i tym, o którym najczęściej się słyszy w przypadku tej dziedziny nauki jest Python. Mimo to, w innych językach wciąż można znaleźć liczne rozwiązania, które pod względem wydajności mogą dorównywać wcześniej wspomnianemu językowi. Ponieważ brakuje prac naukowych porównujących narzędzia w różnych językach programowania pod kątem uczenia maszynowego, to w niniejszym artykule zbadano to zagadnienie.

2. Cel badań i hipotezy

Celem artykułu jest przeanalizowanie i porównanie narzędzi uczenia maszynowego w wybranych językach programowania z podziałem na języki interpretowane, jak i kompilowane. W ramach tego tematu zostanie zbadana wydajność, szybkość, złożoność istniejących narzędzi w wybranych językach pod kątem uczenia maszynowego przy użyciu tych samych algorytmów oraz na tych samych zbiorach danych.

W artykule postawiono następuje hipotezy badawcze. Spełnienie większej ich części będzie równoznaczne z potwierdzeniem, że istnieją przypadki, w których pod względem wydajnościowym, czasowym, lub funkcjonalnym, zdecydowanie lepiej jest skorzystać z istniejących narzędzi w językach programowania o interpretowanym kodzie:

H1: *Biblioteki interpretowanych języków programowania mają najkrótszy czas uczenia modeli,*

H2: *Liczba linii kodu z wykorzystaniem bibliotek języków interpretowanych jest mniejsza niż w językach kompilowanych,*

H3: *Modele wyuczone dzięki narzędziom w interpretowanych językach programowania mają większą dokładność niż te stworzone w tych, które są kompilowane.*

3. Przegląd literatury

Poniżej przedstawiono przegląd literatury związanej z prezentowanym zagadnieniem. Rozwój dziedziny sztucznej inteligencji jest bardzo dynamiczny w ostatnich latach. Istnieje mało artykułów i prac opisujących narzędzia uczenia maszynowego. Dostępna literatura skupia

się głównie na przedstawieniu konkretnych pakietów, bibliotek bez porównania ich wydajności lub możliwości względem innych bibliotek programistycznych w tym samym lub innym języku programowania.

Podczas analizy artykułów dotyczących porównań języków programowania można natrafić jedynie na porównanie maksymalnie dwóch lub trzech najpopularniejszych języków. Istniejące prace nie są najlepszej jakości, porównują tylko czas kompilacji i zużycie pamięci. Nie zawierają natomiast informacji na temat wydajności algorytmów i modeli uczenia maszynowego [1], co jest kluczowe w przypadku tego artykułu, ponieważ porównują jedynie języki, a nie narzędzia służące tworzeniu rozwiązań z zakresu uczenia maszynowego.

Większość artykułów skupia się na przedstawieniu pojedynczego narzędzia w danym języku, biblioteki lub innej platformy, które umożliwiają realizację zadań związanych z uczeniem maszynowym. Jednym z przykładów jest Weka [2, 3] - rozwiązanie bazowane na Javie, które nie wymaga od użytkownika programowania. Wszelkie algorytmy i modele można ustawić w specjalnie do tego przeznaczonej aplikacji, i oczekiwać na rezultaty. W wielu przypadkach jednak nadal można wykorzystać klasy z tej biblioteki w tworzonym kodzie. Innymi popularnymi rozwiązaniami w tym języku są JavaML [4] skupiający się w głównej mierze na algorytmach grupowania i klasteryzacji oraz biblioteka Encog [5], posiadająca algorytmy regresji, klasyfikacji, i klasteryzacji. W trakcie analizy warto również zwrócić uwagę na JIOP [6], ponieważ jest to wysoce zoptymalizowany szkielet programistyczny, który można w pełni konfigurować, a dodatkowo zwraca informację zwrotną w postaci graficznej.

Podczas analizy rozwiązań w języku Julia można natrafić na JuliaSim [7], czyli środowisko symulacji i uczenia maszynowego. Łączy ono ze sobą wiele różnych bibliotek, dzięki czemu zainstalowanie jednego złożonego pakietu pozwala już na pełną pracę z danymi. Język, mimo że mniej popularny, to w wielu pracach jest przedstawiony jako prosty, szybki i idealny do przetwarzania danych [8], co jest bardzo ważne w przypadku sztucznej inteligencji. Naturalnie, można napotkać wiele innych pakietów przeznaczonych do uczenia maszynowego, takich jak MLJ, Knet i Flux [9-11]. Są to jednak mniejsze pakiety, które prezentują własne podejścia do tematu uczenia maszynowego. Spośród wymienionych, Flux jest szczególnie interesujący, ponieważ łączy cechy rozwiązań statycznych, jak i dynamicznych.

Interesujące propozycje można znaleźć także w języku JavaScript. Biblioteki uczenia maszynowego między innymi takie jak ml5.js, magenta.js czy face-api.js używają głównie przetrenowanych modeli, co daje możliwość bezpośredniej interakcji z użytkownikiem na stronach internetowych [12]. Takie podejście ma wiele zalet, między innymi jest to większa prywatność użytkowników, mniejsze zużycie zasobów serwera. Powoduje to mniejsze koszty jego utrzymania, a także wcześniej wspomniana łatwiejsza dostępność dla użytkowników końcowych. Z drugiej strony jest to rozwiązanie o wiele trudniejsze do wdrożenia i kluczowe znaczenie ma w tym przypadku optymalizacja.

Kolejną grupą rozwiązań przeznaczonych do działania w obszarze sztucznej inteligencji są pakiety języka R. Jest to kolejny język przeznaczony w głównej mierze do analizy danych i pozwalający na uczenie modeli. Pakiet iml to rozwiązanie interpretowane, gdzie kod jest wykonywany w czasie rzeczywistym [13]. Jest to popularne narzędzie w tym języku w głównej mierze z tego powodu, że implementuje wszystkie wymagane metody dla uczenia maszynowego. Jednym z najnowszych rozwiązań w tym języku jest pakiet mlr3. Jest to nowa wersja starszego, ponad dziesięcioletniego, pakietu mlr [14], przepisanego w sposób obiektowy, co jest udogodnieniem dla osób preferujących programowanie obiektowe. Autorzy nowszej wersji poprawili również wydajność i rozwiązali wiele problemów obecnych poprzedniej wersji [15].

W oparciu o dostępną literaturę, analiza narzędzi uczenia maszynowego nie mogła pominąć języka Python. Pełni on kluczową rolę w dziedzinie uczenia maszynowego i zarówno profesjonalności, jak i amatorzy używają narzędzi tego języka. Jedna z popularniejszych bibliotek przeznaczonych do uczenia maszynowego jest biblioteka Scikit-learn (sklearn), która charakteryzuje się bardzo dobrą dokumentacją, zawierającą ponad trzysta stron, a także szybkim działaniem [16]. Przy porównaniu grupy sześciu różnych algorytmów, między innymi klasteryzujących, sklearn aż w czterech okazał się liderem w zestawieniu. Analizując poszczególne narzędzia uczenia maszynowego warto również zwrócić uwagę na język, w którym to narzędzie zostało napisane. Pomimo że Python jest najpopularniejszym językiem wykorzystywanym w dziedzinie AI, to bardzo słabo wypada pod względem szybkości obliczeń matematycznych [17], które pełnią kluczową rolę w przypadku uczenia maszynowego. W porównaniu języków programowania C++ prezentuje się znacznie korzystniej, co czyni go polecanym dla doświadczonych programistów ze względu na lepszą wydajność związaną ze statycznym typowaniem [18].

Dodatkowo część bibliotek uczenia maszynowego została stworzona w całości lub w części w innych językach, niż w tych, w których głównie jest wykorzystywana. Najpopularniejszym przykładem jest zdecydowanie Tensorflow [19] umożliwiający szybkie wdrożenie technik uczenia maszynowego i głębokiego uczenia, a którego rdzeń został napisany w C++ i CUDA [20]. Komunikacja odbywa się przez specjalny do tego celu "Wrapper", rodzaj połączenia pomiędzy odrębnymi technologiami. Języki takie jak C lub C++ często są wykorzystywane w kluczowych fragmentach różnych bibliotek, w których szczególnie ważna jest wydajność działania. Kolejnym przykładem jednego rozwiązania wykorzystywanego w kilku językach programowania jest pakiet RWeka [21], którego rdzeń, czyli Weka, był już wcześniej omówiony. Stanowi to przede wszystkim udogodnienie dla programistów, którzy zyskują zwiększoną dostępność do narzędzi analizy danych oraz zwiększoną elastyczność. Umożliwia to korzystanie z szerokiego zakresu rozwiązań, a nie tylko tych dostępnych domyślnie w danych środowiskach programistycznych.

Wraz z dalszym rozwojem sztucznej inteligencji jest szansa na powstanie nowego języka programowania [22], który będzie utworzony specjalnie dla tej dziedziny. Łączyłby on najlepsze cechy wszystkich dostępnych narzędzi, pozostając przy tym prosty w użyciu.

4. Metoda badawcza

Podczas opracowywania metody badawczej wybrano algorytmy, które były przedmiotem dalszej analizy. W tym procesie wyodrębniono dwie grupy algorytmów na podstawie ich sposobu uczenia. Jedną z grup stanowią algorytmy, które podczas uczenia wykorzystują zbiór treningowy zawierający rzeczywiste wyniki rozwiązania danego problemu. Druga grupa to z kolei algorytmy nienadzorowane, czyli takie, które dla danych wejściowych nie posiadają etykiet, pracują jedynie na surowych danych. Wszelkie powiązania algorytm musi znaleźć sam. Finalny podział typów algorytmów wraz z nazwą wykorzystanego algorytmu w każdej z grup prezentuje się następująco:

1. Uczenie nadzorowane:
   a) Algorytm rozwiązujący problem regresji (regresja liniowa),
   b) Algorytm rozwiązujący problem klasyfikacji (SVM).
2. Uczenie nienadzorowane:
   a) Algorytm klasteryzacji (k-średnich).

4.1. Stanowisko badawcze

Porównanie ze sobą wybranych języków programowania zostało przeprowadzone na podstawie stworzonego kodu spełniającego te same założenia. Kod ten został napisany z użyciem istniejących rozwiązań do uczenia maszynowego w wybranych językach. Wszystkie eksperymenty zostały przeprowadzone na tym samym stanowisku badawczym, którym był komputer stacjonarny przy pełnym wykorzystaniu swoich podzespołów. Oznacza to, że wszystkie inne programy na stanowisku zostały zamknięte, a środowisko programistyczne miało ustawione maksymalne dostępne ustawienia zużycia pamięci. Tabela 1 przedstawia zamontowane na stanowisku badawczym podzespoły.

Tabela 1: Tabela podzespołów stanowiska badawczego

Podzespół	Model
Procesor	AMD Ryzen 5 3600 (6 rdzeni, 12 wątków)
Karta graficzna	NVIDIA GeForce GTX 1660
Ram	16GB DDR4 (taktowanie 3200MHz)
Dysk twardy	Samsung SSD 970 EVO Plus 500GB
System operacyjny	Windows 11 Pro

4.2. Zbierane wyniki

Celem eksperymentów badawczych było zebranie wyników istniejących rozwiązań z zakresu uczenia maszynowego. Zbierane wyniki dotyczyły:

1. Skuteczności modelu - wyniki dla danego modelu na podstawie dobranych metryk dla algorytmów przedstawiającej dokładność wyuczonego modelu. Wszystkie stworzone modele w obrębie danego algorytmu uczyły się na podstawie tej samej liczby epok, aby zapewnić obiektywność pomiarów,
2. Czasu trenowania modelu - czas potrzebny na wyuczenie modelu wraz z czasem potrzebnym na wczytanie i wstępne przetworzenie danych. Skupiono się jedynie na rzeczywistym czasie działania programu, a nie na czasie potrzebnym do jego kompilacji.
3. Liczba linii kodu programu - całościowa liczba linii kodu potrzebna na wczytanie i naukę modelu. Pominięto części programu odpowiedzialne za wczytanie zewnętrznych bibliotek, mierzenie czasu działania, jak i wyświetlanie potrzebnych wyników. Aby uzyskać bardziej rzetelne i wiarygodne wyniki, w każdym z badanych narzędzi programistycznych zastosowano zasady pisania czystego kodu, a dodatkowo nie liczono pustych linii.

W przypadku algorytmu k-średnich, należącego do grupy algorytmów klasteryzacji, zdecydowano się zbierać wyniki przy podziale zbioru na sześć klastrów.

W trakcie analizy porównawczej istniejących narzędzi w interpretowanych i kompilowanych językach programowania zdecydowano się skoncentrować na najlepszym wyniku dla wybranej biblioteki w danym języku programowania w ciągu dwudziestu prób. Takie podejście pozwala oszacować maksymalną wydajność testowanych bibliotek, co jest kluczowe dla oceny ich szczytowych możliwości w idealnych warunkach. Wybranie najlepszych pomiarów na podstawie osiągniętych skuteczności modeli minimalizuje wpływ losowych zakłóceń, które mogą wynikać z różnych czynników zewnętrznych, takich jak inne procesy działające w tle. Dzięki temu otrzymano bardziej uczciwe porównanie, eliminując nieprzewidziane zmienności i wahania wyników.

4.3. Metryki

Na podstawie typów badanych algorytmów dobrano metryki służące do ich oceniania. Jednym z najpopularniejszych formatów prezentowania wyników wśród wszystkich dostępnych dla algorytmu klasyfikacji jest wynik procentowy, określający wydajność modelu [23]. Konkretnie, jest to stosunek liczby poprawnie sklasyfikowanych obserwacji do całkowitej ich liczby w zbadanej próbce.

W przypadku regresji za skuteczność badania ma służyć miara błędu rzeczywistych pomiarów od tych przewidywanych. Najczęściej używaną i dostarczającą najwięcej informacji metryką ukazującą skuteczność wyuczonych modeli jest między innymi błąd średniokwadratowy (MSE), który mierzy rozproszenie danych od linii regresji. Zalecana jest jak najmniejsza wartość tej metryki [24]. Kolejną metryką jest średni bezwzględny błąd procentowy (MAPE), który ukazuje średnią wielkość błędów przewidywań wyuczonego modelu.

Rozwiązania nienadzorowane nie pozwalają na bezpośrednią ocenę skuteczności stworzonego rozwiązania. Nie można stworzyć takiej statystyki, ponieważ w takich zbiorach danych brakuje etykiety z oczekiwanymi wynikami grupowań próbek badawczych. W modelach segmentacyjnych jakim jest między innymi model

klasteryzacji za ocenę jakości służy najczęściej inercja, czyli suma kwadratów pomiaru odległości między każdym punktem danych, a jego centroidą w każdym wytyczonym skupieniu. Jest to intuicyjna miara oceny jakości segmentacji i z tego powodu została ona również wykorzystana w trakcie tego badania. Kolejną obiektywną miarą jest współczynnik Silhouette'a, który wyznacza podobieństwo klastrów poprzez obliczenie średniej odległości obserwacji wewnątrz każdego klastra oraz średniego dystansu do najbliższego innego klastra [25].

4.4. Wersje analizowanych narzędzi

W trakcie przeprowadzania eksperymentów zwrócono również uwagę na wersje wykorzystywanego oprogramowania, a także bibliotek, pakietów wykorzystywanych dla omawianych badań i zaprezentowano zbiorczą informację w Tabeli 2.

Tabela 2: Tabela wersji, narzędzi, bibliotek i pakietów

Narzędzie	Wykorzystywany język programowania dla danego narzędzia	Wersja narzędzia
IntelliJ IDEA Ultimate	Java	2023.3.5
Oracle OpenJDK	Java	21.0.2
Encog	Java	3.4
weka-stable	Java	3.8.6
Python Interpreter	Python	3.12.2
scikit-learn (sklearn)	Python	1.4.1.post1
Visual Studio Code	Python, Julia	1.87.2
Julia	Julia	1.10.2
MLJ.jl	Julia	0.16.5
Lathe	Julia	0.1.6
GNU R	R	4.3.3
R Studio	R	26.2.4
cluster	R	2.1.6
mlr3	R	0.18.0
Visual Studio 2022	C#	17.9.4
ML.NET	C#	3.0.1
.NET	C#	8.0

4.5. Zbiory danych

Dla każdego typu algorytmu wybrano i opisano zbiory, które będą użyte w dalszej części artykułu. Zwrócono uwagę na dostępność, a także możliwość porównania ich wyników.

1. Algorytm regresji – dla tego algorytmu zdecydowano się na zbiór „Auto-MPG", który jest dostępny między innymi w repozytorium uczenia maszynowego Uniwersytetu Kalifornijskiego w Irvine [26]. Opisuje on właściwości samochodów i ich osiągi. Wynikiem regresji liniowej będzie liczba określająca spalanie danego samochodu w galonach osobno dla każdego z 398 danych samochodów występujących w zbiorze,
2. Algorytm klasyfikacji – w tym przypadku za zbiór testowy posłuży zbiór danych dotyczących możliwej cukrzycy pacjentek, a zawierający ich wyniki medyczne, między innymi część wyników morfologii krwi. Rezultatem badań będzie określenie dla każdego wiersza danych przewidywania odnośnie występowania cukrzycy u pacjentki. Zbiór ten jest dostępny na platformie Kaggle [27] i posiada aż 2768 wierszy danych,
3. Algorytm klasteryzacji – ostatnim rozpatrywanym zbiorem danych będzie ten o nazwie „Mall Customer Segmentation Data" ze strony internetowej Kaggle [28]. Posiada on etykiety dotyczące wieku, rocznych dochodów, jak i oceny konsumenta na podstawie wydatków w centrum handlowym. Wynikiem badania będzie podział danych na sześć zbiorów, a następnie ocena dokonanego przez algorytm uporządkowania wszystkich z 200 dostępnych rekordów zbioru.

4.6. Implementacja

Na listingach 1-3 przedstawiono implementację uczenia modelu oraz odczytu wyników w testowanych językach programowania. Listing 1 pokazuje kod odpowiedzialny za regresję z wykorzystaniem biblioteki sklearn w języku Python. Kod umieszczony na Listingu 2 realizuje zadanie klasyfikacji przy użyciu pakietu mlr3 w języku R. Na Listingu 3 ukazano kod odpowiedzialny za klasteryzację z wykorzystaniem pakietu Lathe w języku Julia.

Listing 1: Kod korzystający z biblioteki sklearn w języku Python realizujący algorytm regresji

```
df = pd.read_csv("regresja.csv")
df = df.drop("car name", axis=1).replace('?', np.nan)
df['horsepower'] = df['horsepower'].astype('float64')
mean_fill = lambda x: x.fillna(x.mean())
df = df.apply(mean_fill,axis=0)
df = pd.get_dummies(df, columns=['origin'])
x = df.drop(['mpg'], axis=1)
y = df[['mpg']]
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.30, random_state=1)
regression_model = LinearRegression()
regression_model.fit(x_train, y_train)
y_pred = regression_model.predict(x_test)
```

Listing 2: Kod korzystający z pakietu mlr3 w języku R realizujący algorytm klasyfikacji

```
data <- read.csv("./klasyfikacja.csv", na.strings = "?")
data <- data[, -c(1, 7)]
for(i in 1:ncol(data)){
  data[is.na(data[,i]), i] <- mean(data[,i], na.rm = TRUE)
}
my_task <- as_task_classif(data, target = "Outcome")
split = partition(my_task, ratio = 0.7)
lrn = lrn("classif.svm")
lrn$train(my_task, split$train)
prediction = lrn$predict(my_task, split$test)
```

Listing 3: Kod korzystający z pakietu Lathe w języku Julia realizujący algorytm klasteryzacji

```
df = select!(DataFrame(CSV.File("./klasteryzacja.csv")), Not([:CustomerID, :Gender]))
df = collect(Matrix(df)')
predicted = kmeans(df, 6)
silh = clustering_quality.(Ref(df), hcat(predicted), quality_index = :silhouettes)
inertia = calculate_inertia(df, predicted.centers)
```

5. Wyniki

Wszystkie wyniki z testowanych narzędzi zebrano, a następnie zaprezentowano poszczególne czasy nauki modeli oraz wymaganą minimalną liczbę linii kodu dla algorytmów w danych językach. Przedstawiono również porównanie metryk ich dotyczących dla każdej implementacji algorytmu z danej grupy. W tabelach zdecydowano się prezentować jedynie najlepsze osiągnięte czasy bibliotek z każdego języka.

Wszystkie otrzymane wyniki w trakcie badania obejmujące czas działania algorytmu regresji liniowej, liczbę

linii kodu potrzebną do uruchomienia programu oraz metryki, takie jak błąd średniokwadratowy i średni bezwzględny błąd procentowy, zostały zebrane w Tabeli 3.

Tabela 3: Tabela wyników algorytmu regresji

Badana statystyka / Język (Biblioteka)	MSE	MAPE (%)	Czas (s)	Linie kodu
Java (Weka)	9,498	10,287	0,376	22
Python (sklearn)	9,17	11,152	0,184	12
C# (ML.NET)	11,643	11,578	0,337	116
Julia (Lathe)	10,295	11,62	0,438	24
R (mlr3)	11,942	12,552	0,225	23

Tabela 4 ukazuje metrykę dotyczącą wydajności modeli, potrzebną liczbę linii kodu do wczytania i wyuczenia modeli, jak i szybkość działania programów wybranych języków programowania w przypadku algorytmu SVM.

Tabela 4: Tabela wyników algorytmu klasyfikacji

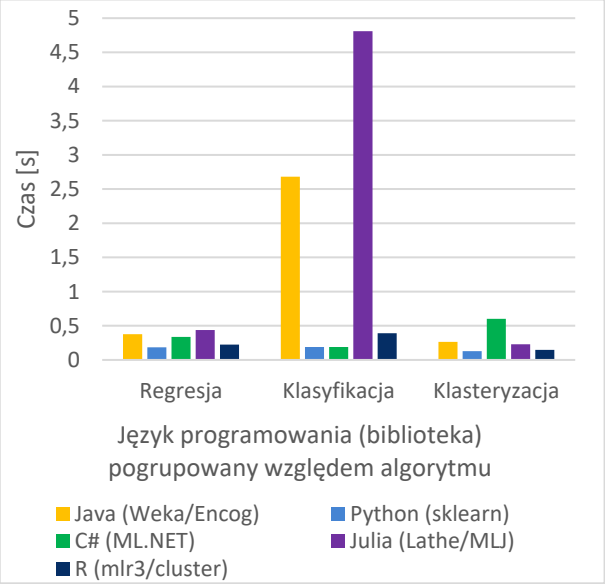
Badana statystyka / Język (Biblioteka)	Wydajność (%)	Czas (s)	Linie kodu
Java (Encog)	83,01	2,68	36
Python (sklearn)	76,90	0,19	15
C# (ML.NET)	77,95	0,19	120
Julia (MLJ)	77,22	4,81	18
R (mlr3)	82,67	0,39	10

W Tabeli 5 przedstawiono wyniki dla omawianych narzędzi w pięciu językach zgodnie z opisanymi wcześniej metrykami odnośnie inercji, jak i współczynnika Silhouette'a dla algorytmu k-średnich.

Tabela 5: Tabela wyników algorytmu klasteryzacji

Badana statystyka/ Język (Biblioteka)	Inercja	Współczynnik Silhouette'a	Czas (s)	Linie kodu
Java (Weka)	60167	0,455	0,265	14
Python (sklearn)	58337	0,451	0,129	5
C# (ML.NET)	58300	0,452	0,601	32
Julia (Lathe)	58300	0,452	0,229	5
R (cluster)	58300	0,452	0,147	4

Dla metryki czasu działania programów zdecydowano się również na przedstawienie wyników za pomocą wykresu kolumnowego, pogrupowanego według realizowanych algorytmów. Wykres ten został zaprezentowany na Rysunku 1.



Rysunek 1: Wykres czasu działania narzędzi w wybranych językach programowania pogrupowanych według algorytmu.

6. Wnioski

W przypadku badania algorytmu regresji liniowej najniższe miary średniego błędu kwadratowego, jak i średniego bezwzględnego błędu procentowego uzyskano dla bibliotek sklearn w języku Python i Weka w języku Java. Różnice w uzyskanych czasach były na tyle znaczące, że można je było sklasyfikować ze względu na typ języka programowania, dla którego została stworzona dana biblioteka. Czas działania programów opartych o biblioteki języka R, jak i języka Python był zdecydowanie krótszy, co najmniej o jedną trzecią całkowitej wartości w porównaniu z dostępnymi narzędziami w językach kompilowanych.

Wyniki dla algorytmu klasyfikacji pokazują, że to właśnie Encog, czyli narzędzie języka kompilowanego, napisanego dla języka Java, uzyskało najlepszy wynik pod względem wydajności. Nieznacznie gorzej wypadło wymienione w tym zestawieniu narzędzie mlr3 dostępne w języku R, który jest językiem interpretowanym. Pozostałe badane biblioteki w omawianych językach uzyskały wynik na poziomie około 6% niższym niż te z czołówki. W przypadku badanego czasu widać, że biblioteki dla języków interpretowanych wykonują swoje zadanie w krótszym czasie niż pozostałe. Co warte zaznaczenia, algorytm stworzony przy wykorzystaniu ML.NET w języku C# potrzebował tyle samo czasu co rozwiązanie napisane z użyciem biblioteki sklearn z języka Python, pomimo że ten pierwszy wymieniony język jest również językiem kompilowanym.

W trakcie badania algorytmu nienadzorowanego uczenia maszynowego dla klasteryzacji otrzymano zbliżone do siebie wyniki. W trzech z pięciu narzędzi, wyniki zarówno dla inercji, jak i współczynnika Silhouette'a były takie same. Oznacza to, że algorytm k-średnich, zwany algorytmem centroidów, w identyczny sposób dokonał podziału zbioru danych na założone sześć

klastrów. W przypadku badania algorytmu klasteryzacji, czas również można było zakwalifikować do dwóch grup, a różnica pomiędzy najkrótszym, a najdłuższym czasem działania programu wyniosła prawie pół sekundy.

Ostatnią badaną cechą we wszystkich rodzajach algorytmów była liczba linii kodu potrzebna na wczytanie danych z pliku o formacie CSV, wstępne przetworzenie ich, a następnie nauczenie modelu na ich podstawie. W każdym z trzech badań najlepszymi okazały się narzędzia w językach interpretowanych. Dwukrotnie były to te w języku R – biblioteki cluster i mlr3, i raz biblioteka sklearn z języka Python. Warty zauważenia jest tu fakt, że w porównaniu z narzędziem ML.NET w C#, a narzędziami w językach interpretowanych występuje nawet dwunastokrotna różnica w przypadku liczby linii kodu. Spowodowane jest to tym, że język Python charakteryzuje się znacznie wyższym poziomem abstrakcji oraz bardziej zwięzłą składnią. Z kolei w C# często wymagane jest tworzenie bardziej szczegółowych instrukcji oraz jawne definiowanie wielu operacji, co prowadzi do pisania większej liczby linii w celu osiągnięcia podobnych rezultatów.

7. Podsumowanie

Celem artykułu była analiza porównawcza narzędzi uczenia maszynowego w językach interpretowanych, jak i kompilowanych. Aby tego dokonać, utworzono realizujące te same zadania regresji, klasyfikacji i klasteryzacji implementacje kodu w narzędziach z grupy pięciu wybranych języków programowania. Łącznie zostało stworzonych aż piętnaście takich wykonywalnych programów, po trzy dla każdego języka, a po pięć dla każdego typu algorytmu - regresji, klasyfikacji i klasteryzacji.

Porównując wyniki bibliotek Weka, Encog, ML.NET, MLJ, Lathe z języków Java, C# i Julia z najpopularniejszymi narzędziami w językach w tej kategorii, czyli z sklearn w Pythonie oraz mlr3 w R można zauważyć, że oscylują na tym samym lub niewiele gorszym poziomie pod względem wydajności uczenia modeli. Spowodowane jest to tym, że pomimo utworzenia kodu w różnych językach programowania, te same algorytmy bazują na tym samym działaniu i sposobie uczenia każdej epoki. W przypadku regresji najlepszym narzędziem okazał się sklearn dostępny w języku Python. Scenariusz badawczy związany z klasyfikacją danych wyłonił jako lidera bibliotekę Encog języka Java, a w przypadku klasteryzacji zarówno narzędzia języków z grupy interpretowanych, jak i kompilowanych uzyskały jednakowy, najlepszy wynik. Zatem hipoteza dotycząca dokładności wyuczonych modeli nie została potwierdzona.

Kolejną dostrzegalną różnicą w wynikach jest krótszy czas spowodowany częściowym wykorzystaniem języków, takich jak C lub C++ przy tworzeniu bibliotek języków interpretowanych, które to zwiększają wydajność w porównaniu z bibliotekami pisanymi w całości w innych językach programowania, jak na przykład MLJ czy Encog. Ostatnią znaczącą różnicą była liczba linii kodu potrzebna do implementacji algorytmu w konkretnym języku programowania. Narzędzia w językach interpretowanych w każdym badaniu znajdowały się na najlepszej pozycji, ponieważ potrzebowały ich najmniej. Jest to ważne z tego względu, że długość programów wpływa na czas potrzebny na jego utworzenie, trudność późniejszej analizy, a także koszty związane z pracą programisty nad nim.

Dodatkowo popularne interpretowane języki, które są głównie używane w przypadku zadań związanych ze sztuczną inteligencją posiadają więcej bibliotek zawierających dodatkowe rodzaje algorytmów, ich typów, a także metryk. W powyższych badaniach ten element nie był brany pod uwagę, jednak warto o nim wspomnieć. Algorytmy dla badań zostały wyselekcjonowane tak, aby możliwa była ich implementacja w każdym z języków. W przypadku braku gotowych funkcji do obliczania metryk, były one ręcznie tworzone według opisujących ich wzorów. Jeśli zdecydowano by się na implementację złożonych rozwiązań, mogłoby się okazać tak, że w części języków byłoby to niewykonalne.

Podsumowując całość artykułu, uzyskane wyniki, a także wystosowane na ich podstawie wnioski, można stwierdzić, że większa część wszystkich postawionych hipotez została spełniona. Dwie hipotezy zostały potwierdzone – pierwsza dotycząca czasu działania programu, jak i druga dotycząca liczby linii kodu. Jedna związana z wydajnością wytrenowanych modeli została niepotwierdzona, ponieważ wyniki w obu przypadkach były ze sobą porównywalne. Takie spełnienie hipotez jednoznacznie potwierdza tezę, że biblioteki języków interpretowanych są najskuteczniejsze w tworzeniu rozwiązań uczenia maszynowego. Uwzględniono przy tym czas, liczbę linii kodu oraz wydajność wyuczonych modeli. Dostępne narzędzia w językach kompilowanych w przyszłości mogą stać się dobrą alternatywą do tworzenia rozwiązań z zakresu uczenia maszynowego, jak i całej sztucznej inteligencji. Ich popularność w dziedzinie uczenia maszynowego prawdopodobnie będzie stale rosnąć, jednak na podstawie obecnych wyników, nie mogą się aktualnie równać z już sprawdzonymi technologiami wykonującymi kod przy pomocy interpretera, do których należą istniejące rozwiązania przeznaczone dla języków Python i R.

Literatura

[1] B. Johnson, A. S. Chandran, Comparison between Python, Java and R programming language in machine learning, International Research Journal of Modernization in Engineering Technology and Science 3(6) (2021) 1–6.

[2] M. Wickham, Practical Java Machine Learning, Apress, Irving, 2018.

[3] I. H. Witten, E. Frank, L. E. Trigg, M. A. Hall, G. Holmes, S. J. Cunningham, Weka: Practical machine learning tools and techniques with Java implementations, Working Paper, The University of Waikato, Hamilton, 1999.

[4] T. Abeel, Y. Van de Peer, Y. Saeys, Java-ML: A Machine Learning Library, Journal of Machine Learning Research 10(34) (2009) 931–934, https://dl.acm.org/doi/10.5555/1577069.1577103.

[5] J. Heaton, Encog: Library of Interchangeable Machine Learning Models for Java and C#, Journal of Machine Learning Research 16(36) (2015) 1243–1247, https://doi.org/10.48550/arXiv.1506.04776.

[6] L. I. Hatledal, F. Sanfilippo, H. Zhang, JIOP: A Java Intelligent Optimisation and Machine Learning Framework, Proceedings of the European Conference on Modelling and Simulation (2014) 1-7, http://dx.doi.org/10.7148/2014-0101.

[7] C. Rackauckas, R. Anantharaman, A. Edelman, S. Gowda, M. Gwozdz, A. Jain, C. Laughman, Y. Ma, F. Martinuzzi, A. Pal, U. Rajput, E. Saba, V. B. Shah, Composing Modeling And Simulation With Machine Learning In Julia, Proceedings of the Annual Modeling and Simulation Conference (ANNSIM) (2022) 1–17, https://doi.org/10.48550/arXiv.2105.05946.

[8] K. Gao, G. Mei, F. Piccialli, S. Cuomo, J. Tu, Z. Huo, Julia language in machine learning: Algorithms, applications, and open issues, Computer Science Review 37 (2020) 1-13, https://doi.org/10.1016/j.cosrev.2020.100254.

[9] A. D. Blaom, F. Kiraly, T. Lienart, Y. Simillides, D. Arenas, S. J. Vollmer, MLJ: A Julia package for composable Machine Learning, Journal of Open Source Software 5(55) (2020) 1-9, https://doi.org/10.21105/joss.02704.

[10] M. Innes, Flux: Elegant machine learning with Julia, Journal of Open Source Software 3(25) (2018) 1, https://doi.org/10.21105/joss.00602.

[11] D. Yuret, Knet: beginning deep learning with 100 lines of Julia, Proceedings of the Machine Learning Systems Workshop at NIPS (2016) 1-7.

[12] H-A. Goh, C-K. Ho, F. S. Abas, Front-end deep learning web apps development and deployment: a review, Applied Intelligence 53(12) (2023) 15923–15945, http://dx.doi.org/10.1007/s10489-022-04278-6.

[13] C. Molnar, G. Casalicchio, B. Bischl, iml: An R package for Interpretable Machine Learning, Journal of Open Source Software 3(26) (2018) 1-2, https://doi.org/10.21105/joss.00786.

[14] M. Lang, M. Binder, J. Richter, P. Schratz, F. Pfisterer, S. Coors, Q. Au, G. Casalicchio, L. Kotthoff, B. Bischl, mlr3: A modern object-oriented machine learning framework in R, Journal of Open Source Software 4(44) (2019) 1-3, https://doi.org/10.21105/joss.01903.

[15] B. Bischl, M. Lang, L. Kotthoff, J. Schiffner, J. Richter, E. Studerus, G. Casalicchio, Z. M. Jones, mlr: Machine Learning in R, Journal of Machine Learning Research 17(170) (2016) 1–5.

[16] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, E. Duchesnay, Scikit-learn: Machine Learning in Python, Journal of Machine Learning Research 12(85) (2012) 2825–2830, https://doi.org/10.48550/arXiv.1201.0490.

[17] S. Ali, S. Qayyum, A Pragmatic Comparison of Four Different Programming Languages, Preprints (2021) 1-14, https://doi.org/10.14293/s2199-1006.1.sor-.pp5rv1o.v1.

[18] F. Zehra, M. Javed, D. Khan, M. Pasha, Comparative Analysis of C++ and Python in Terms of Memory and Time, Preprints (2020) 1-11, http://dx.doi.org/10.20944/preprints202012.0516.v1.

[19] M. Ramchandani, H. Khandere, P. Singh, P. Rajak, N. Suryawanshi, A. S. Jangde, L. Arya, P. Kumar, M. Sahu, Survey: Tensorflow in Machine Learning, Journal of Physics: Conference Series 2273(1) (2022) 1-12, http://dx.doi.org/10.1088/1742-6596/2273/1/012008.

[20] M. N. Gevorkyan, A. V. Demidova, T. S. Demidova, A. A. Sobolev, Review and comparative analysis of machine learning libraries for machine learning, Discrete and Continuous Models and Applied Computational Science 27(4) (2019) 305–315, http://dx.doi.org/10.22363/2658-4670-2019-27-4-305-315.

[21] K. Hornik, C. Buchta, A. Zeileis, Open-source machine learning: R Meets Weka, Computational Statistics 24(2) (2009) 225–232, http://dx.doi.org/10.1007/s00180-008-0119-7.

[22] M. Innes, S. Karpinski, V. B. Shah, D. Barber, P. Stenetorp, T. Besard, J. Bradbury, V. Churavy, S. Danisch, A. Edelman, J. Malmaud, J. Revels, D. Yuret, On Machine Learning and Programming Languages, Proceedings of the SysML Conference (2018) 1-3.

[23] Ž. Đ. Vujović, Classification Model Evaluation Metrics, International Journal of Advanced Computer Science and Applications 12(6) (2021) 599–606, https://dx.doi.org/10.14569/IJACSA.2021.0120670.

[24] M. Hossin, M. N. Sulaiman, A review on evaluation metrics for data classification evaluations, International Journal of Data Mining & Knowledge Management Process 5(2) (2015) 1–11, http://dx.doi.org/10.5121/ijdkp.2015.5201.

[25] K. R. Shahapure, C. Nicholas, Cluster Quality Analysis Using Silhouette Score, Proceedings of the 2020 IEEE 7th International Conference on Data Science and Advanced Analytics (DSAA) (2020) 747–748, http://dx.doi.org/10.1109/DSAA49011.2020.00096.

[26] Data set of cars and their parameters, https://archive.ics.uci.edu/dataset/9/auto+mpg, [15.07.2024].

[27] Data set of possible diabetes in patients, https://www.kaggle.com/datasets/uciml/pima-indians-diabetes-database, [15.07.2024].

[28] Data set of shopping center customers, https://www.kaggle.com/datasets/vjchoudhary7/customer-segmentation-tutorial-in-python, [15.07.2024].