УДК004.43 Python:004.8 **DOI:** 10.31673/2412-9070.2024.050575
А. В. ГОЛОВЧЕНКО[1], аспірант;
ORCID: 0009-0003-6127-0426
О. М. ТКАЛЕНКО[1], канд. техн. наук, доцент;
ORCID: 0000-0001-6313-5138
А. О. ШТАНЬКО[1], студент;
ORCID: 0009-0001-5893-7862
І. Ю. БОБЕНЬ[2], дослідник, NET developer,
ORCID: 0009-0000-9421-1420

[1]Державний університет інформаційно-комунікаційних технологій, Київ
[2] ПАТ Trinetix, Київ

ВИКОРИСТАННЯ МОВИ ПРОГРАМУВАННЯ PYTHON ДЛЯ ЗАДАЧ ШТУЧНОГО ІНТЕЛЕКТУ

Використання мови програмування Python для задач штучного інтелекту є корисним та актуальним напрямом, маючи простоту та зручність синтаксису, велику кількість бібліотек, гнучкість та масштабованість, широкий спектр застосувань. Ці фактори роблять Python оптимальним вибором для розробки проєктів у сфері штучного інтелекту, забезпечуючи швидкість розробки, доступ до потужних інструментів та можливість працювати з сучасними методиками.

У роботі досліджені популярні бібліотеки Python, такі як NumPy, Pandas, Matplotlib, Seaborn, розглянуті приклади використання машинного навчання з використанням алгоритму Naive Bayes та методу опорних векторів Support Vector Machine (SVM), визначено точність моделей машинного навчання на тестовому наборі даних. Визначено ефективність машинного навчання для класифікації електронних листів. Надано рекомендації щодо підвищення ефективності використання вищевказаних підходів.

Ключові слова: *штучний інтелект, Python, машинне навчання, класифікація, електронні листи, Naive Bayes, Support Vector Machine (SVM).*

Вступ

Штучний інтелект (ШІ) широко використовується у багатьох сферах, включаючи аналіз даних та обробку природної мови. Мова програмування Python є однією з найбільш популярних мов для розробки алгоритмів ШІ. У світі, де технології стрімко розвиваються, використання машинного навчання стає необхідністю для вирішення різноманітних завдань, особливо у сфері обробки даних. Python, який є однією з найпопулярніших мов програмування, став майже стандартом у галузі машинного навчання. Його зручність та багатофункціональність дозволяють розробникам швидко та ефективно створювати та вдосконалювати алгоритми машинного навчання [1-3].

Python є мовою програмування, яка стала популярною серед розробників завдяки своїй простоті та ефективності. Вона використовується у багатьох сферах від веб-розробки до штучного інтелекту, завдяки чому є однією з найбільш розповсюджених мов у світі програмування.

Популярними бібліотеками Python є такі бібліотеки як NumPy, Pandas, Matplotlib, Seaborn та багато інших.

Бібліотека NumPy забезпечує роботу з числовими даними, спрощує роботу з числовими даними, такими як масиви та матриці. Вона надає швидкі та ефективні функції для математичних операцій, лінійної алгебри, генерації випадкових чисел. NumPy є незамінним інструментом для наукових обчислень та аналізу даних.

Бібліотека Pandas забезпечує роботу з табличними даними. Pandas - це бібліотека для аналізу та обробки даних у форматі табличних структур, таких як DataFrame та Series. Вона дозво-

ляє зручно завантажувати, фільтрувати, сортувати, агрегувати та візуалізувати дані. Pandas забезпечує потужні інструменти для роботи з великими обсягами даних у Python.

Бібліотеки Matplotlib та Seaborn використовуються для візуалізації даних. Matplotlib та Seaborn - це бібліотеки для створення графіків та візуалізації даних. Вони надають можливості для побудови різноманітних графіків, діаграм, теплових карт та інших візуальних представлень даних. Ці бібліотеки допомагають зрозуміти та відобразити дані у зручному та зрозумілому форматі. У статті будуть розглянуті приклади використання машинного навчання для класифікації електронних листів на "спам" та "не спам" для розділення важливих повідомлень від не-бажаної реклами з використанням Python та його бібліотек.

Основна частина

Для роботи буде використовуватися бібліотека Pandas, яка призначена для аналізу та обробки даних, для класифікації електронних листів на "спам" та "не спам". За основу буде взято алгоритм Naive Bayes та метод опорних векторів Support Vector Machine (SVM). Вони є популярними методами машинного навчання, які використовуються для класифікації та регресії у задачах аналізу даних [2-7].

Алгоритм Naive Bayes базується на теоремі Байєса та вважає, що всі ознаки у наборі даних незалежні одна від одної. Цей алгоритм використовує ймовірнісний підхід для класифікації об'єктів. Naive Bayes часто використовується у задачах класифікації текстів (наприклад, спам-фільтри в електронній пошті), аналізі настроїв (sentiment analysis), медичних діагнозах та інших сферах, де важлива точність та ефективність в умовах обмеженого обсягу даних.

Метод опорних векторів (SVM) є методом надзвичайної потужності, який забезпечує ефективну класифікацію та регресію шляхом знаходження оптимальної границі розділення між класами. Цей метод шукає гіперплощину, яка максимізує відстань між об'єктами різних класів. SVM застосовується у багатьох областях, таких як розпізнавання образів, біомедичні дослідження, прогнозува-ння фінансових показ-ників, класифікація текс-тів та інші задачі, де важлива точність та здатність до роботи з великими обсягами даних. Naive Bayes працює добре з невеликими обсягами даних та припускається, що ознаки незалежні. Він швидко навчається та добре підходить для текстових даних. SVM ефективний для вирішення складних задач класифікації та регресії, зокрема у випадках, коли дані мають складну структуру та велику кількість ознак. Обидва алгоритми є потужними інструментами у машинному навчанні, їх вибір залежить від конкретної задачі та особливостей даних. Розглянемо два методи використання машинного навчання для класифікації електрон-них листів через Python в Visual Studio Code [8-10].

```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

# Завантаження даних
data = pd.read_csv('email_dataset_100.csv', encoding='utf-8')

# Розділення даних на тренувальні та тестові набори
X_train, X_test, y_train, y_test = train_test_split(data['text'], data['label'], test_size=0.8, random_state=42)

# Перетворення текста в TF-IDF числові дані
vectorizer = TfidfVectorizer()
X_train_vectorized = vectorizer.fit_transform(X_train)
X_test_vectorized = vectorizer.transform(X_test)

# Побудова моделі
model = LogisticRegression(max_iter=1000)
model.fit(X_train_vectorized, y_train)

# Прогнозування на тестовому наборі
y_pred = model.predict(X_test_vectorized)

# Оцінка точності моделі
accuracy = accuracy_score(y_test, y_pred)
print("Точність моделі на тестовому наборі:", accuracy)

# Створення DataFrame для порівняння реальних та передбачених міток
comparison = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})

# Зберігання всіх даних в CSV файл
comparison.to_csv('model_comparison.csv', index=False)
```

Рис. 1. Класифікація електронних листів за допомогою Naive Bayes

Точність моделі на тестовому наборі даних складає *0.5375*. Файл comparison.csv вивів 43 мітки, де реальні та передбачені мітки збігаються. Тому, точність моделі на 80 тестових записах - 43/80 = 0.5375.

Основною особливістю Naive Bayes є припущення про незалежність ознак (features). Це означає, що всі ознаки вважаються незалежними одна від одної, що, як правило, не відповідає дійсності в реальних даних. Проте - це спрощує обчислення й робить алгоритм дуже швидким і простим у реалізації. Алгоритм використовує теорему Байєса для обчислення ймовірності того, що зразок належить до певного класу. Формула теореми Байєса має вигляд:

$$P(C|X) = \frac{P(X|C) \cdot P(C)}{P(X)} \quad , \tag{1}$$

де: *P(C/X)* - ймовірність того, що зразок належить до класу *C*, за умови спостереження ознак *X*;

P(X/C) - ймовірність спостереження ознак *X*, за умови, що зразок належить до класу *C*;

P(C) - апріорна ймовірність класу *C*;

P(X) - ймовірність спостереження ознак *X*.

Завдяки наївному припущенню незалежності, обчислення ймовірностей є простим і швидким. Це робить Naive Bayes ефективним навіть для великих наборів даних. Наївний Байєс добре працює навіть на малих обсягах даних, оскільки не вимагає складного моделювання або великої кількості параметрів для навчання. Оскільки алгоритм припускає незалежність ознак, він менш схильний до проблеми багатоколінеарності (високої кореляції між ознаками), яка може впливати на інші моделі, такі як регресія.

```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

# Завантаження даних
data = pd.read_csv('email_dataset_100.csv', encoding='utf-8')

# Розділення даних на тренувальні та тестові набори
X_train, X_test, y_train, y_test = train_test_split(data['text'], data['label'], test_size=0.8, random_state=42)

# Перетворення текста в TF-IDF числові дані
vectorizer = TfidfVectorizer()
X_train_vectorized = vectorizer.fit_transform(X_train)
X_test_vectorized = vectorizer.transform(X_test)

# Побудова моделі
model = SVC(kernel='rbf', gamma='scale')
model.fit(X_train_vectorized, y_train)

# Прогнозування на тестовому наборі
y_pred = model.predict(X_test_vectorized)

# Оцінка точності моделі
accuracy = accuracy_score(y_test, y_pred)
print("Точність моделі на тестовому наборі:", accuracy)

# Створення DataFrame для порівняння реальних та передбачених міток
comparison = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})

# Зберігання всіх даних в CSV файл
comparison.to_csv('model_comparison_svm_rbf.csv', index=False)
```

Рис. 2. Класифікація електронних листів за допомогою Support Vector Machine (SVM)

Точність моделі на тестовому наборі даних складає 0.55. Файл comparison.csv вивів 43 мітки, де реальні та передбачені мітки збігаються. Тому, точність моделі на 80 тестових записах - 44/80 = 0.55.

SVM шукає гіперплощину, яка максимально розділяє дані різних класів у багатовимірному просторі. Метою є знайти таку гіперплощину, яка має найбільшу відстань (маржу) до найближчих точок з обох класів. Ці найближчі точки називаються опорними векторами, і саме вони визначають положення гіперплощини. Основна ідея SVM полягає в тому, щоб максимізувати відстань між гіперплощиною та найближчими точками обох класів. Цей підхід допомагає мінімізувати помилки класифікації та підвищує здатність моделі до узагальнення на нові дані. Метод опорних векторів є потужним інструментом для багатьох практичних задач завдяки своїй здатності до ефективної класифікації та роботи з даними, що мають високу розмірність.

Обидва приклади у задачах машинного навчання використовують різні підходи. Naive Bayes добре підходить для простих задач та має невеликі вимоги до обробки даних. У той час, як SVM є потужнішим і може працювати з більш складними наборами даних, але вимагає більше обчислювальних ресурсів. Вибір конкретного підходу залежить від конкретного завдання та наявних ресурсів.

Точність моделей машинного навчання залежить від багатьох факторів, які можуть впливати на здатність моделі правильно передбачати або класифікувати нові дані. Основними факторами, що впливають на точність моделей є якість та кількість даних, вибір моделі, попередня обробка даних, вибір гіперпараметрів, методи регуляризації, адаптація до нових даних, вибір правильної метрики.

Дані повинні бути якісними, без пропущених значень, шуму або аномалій. Низька якість даних може призвести до погіршення точності моделі. Навчальні дані повинні бути репрезентативними для тієї задачі, яку вирішує модель. Якщо дані не охоплюють всі можливі випадки, модель може не узагальнювати добре. Більша кількість даних зазвичай покращує точність моделі, оскільки модель отримує більше інформації для навчання. Однак це не завжди так, якщо дані мають низьку якість або присутній дисбаланс класів.

Простих моделей може не вистачати для захоплення складних залежностей у даних (недонавчання), тоді як занадто складні моделі можуть перенавчитися (перенавчання) і погано узагальнювати на нових даних. Деякі моделі краще справляються з певними типами даних. Наприклад, лінійні моделі працюють добре на даних, де залежності між змінними є лінійними, тоді як складніші моделі, такі як нейронні мережі, можуть охоплювати нелінійні залежності. У багатьох випадках точність моделей залежить від їх здатності адаптуватися до нових даних, що надходять з часом. Моделі, що не враховують зміни в даних, можуть втратити свою точність.

Точність моделі залежить від гармонійної взаємодії всіх цих факторів. Налаштування та оптимізація моделей є складним ітеративним процесом, що вимагає досвіду, знань і правильного підходу до роботи з даними.

Висновки

Python разом із своїми бібліотеками надає потужні інструменти для розробки програмного забезпечення та аналізу даних. У даній статті ми детально розглянули два приклади використання машинного навчання для класифікації електронних листів та порівняли їхню ефективність. Провівши дослідження, можна зробити висновок, що підвищення ефективності алгоритмів Naive Bayes та методу опорних векторів (SVM) може бути досягнуто шляхом оптимізації різних аспектів роботи моделей, а також за допомогою попередньої обробки даних. Підвищення ефективності Naive Bayes може бути забезпечене шляхом покращення якості даних, оптимізації ознак, збалансування класів, тюнінгу парметрів, а також шляхом використання гібридних моделей. Підвищення ефективності методу опорних векторів (SVM) може бути забезпечене оптимізацією ядер, тюнінгом параметрів регуляризації, попередньою обробкою даних, вибором оптимального набору даних, розширенням простору ознак.

Ці стратегії допоможуть підвищити ефективність алгоритмів Naive Bayes та SVM, зробивши їх більш точними та стійкими до нових даних. Використання алгоритмів класифікації, таких як Naive Bayes та SVM дозволяє автоматизувати процес фільтрації та управління електронною поштою, що робить їх важливими інструментами для організацій та користувачів.

Список літератури

1. Sebastian Raschka and Vahid Mirjalili, "Python Machine Learning", 3, 2019, 770.

2. Andreas C. Müller and Sarah Guido, "Introduction to Machine Learning with Python", 1, 2016, 394.

3. Aurélien Géron "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 2, 2019, 510.

4. Wes McKinney, "Python for Data Analysis", 3rd edition, 2019, 552.

5. Joel Grus, "Data Science from Scratch", 2nd edition, 2019, 406.

6. Jake VanderPlas, "Python Data Science Handbook", 1st edition, 2017, 546.

7. Ethem Alpaydin, "Introduction to Machine Learning", 3rd edition, 2014, 640.

8. Harry Zhang, "The Optimality of Naive Bayes", 1st edition, 2004, 6.

9. Nello Cristianini and John Shawe-Taylor, "An Introduction to Support Vector Machines and Other Kernel-based Learning Methods", 1st edition, 2000, 204.

10. Fabrizio Sebastiani, "Machine Learning in Automated Text Categorization", 1st edition, 2001, 47.

A. Holovchenko, O. Tkalenko, A. Shtanko, I. Boben

USING THE PYTHON PROGRAMMING LANGUAGE FOR ARTIFICIAL INTELLIGENCE TASKS

Using the Python programming language for artificial intelligence (AI) tasks is a highly relevant and practical direction. Python's simplicity, readability, and ease of syntax make it an ideal choice for both beginners and experts. Additionally, Python's extensive library ecosystem, including tools like NumPy, Pandas, Matplotlib, and Seaborn, provides robust support for data analysis, visualization, and machine learning tasks. These libraries offer powerful capabilities for handling large datasets, performing complex statistical analysis, and creating detailed visualizations, making Python not only flexible but also highly scalable for AI projects.

In this study, we explored several popular Python libraries, such as NumPy for numerical computations, Pandas for data manipulation, Matplotlib, and Seaborn for data visualization. We also examined the application of machine learning algorithms, including Naive Bayes and Support Vector Machine (SVM), which are widely used for classification and regression tasks. These algorithms are particularly effective in scenarios such as email classification, where they can be employed to automatically filter and manage emails, distinguishing between spam and non-spam messages.

Moreover, the implementation of these machine learning models was tested on a sample dataset to determine their accuracy in classifying emails. The results demonstrate that Naive Bayes and SVM are both efficient and reliable in processing and classifying textual data. Furthermore, we provided recommendations for enhancing the effectiveness of these algorithms, such as optimizing hyperparameters and using advanced feature engineering techniques to improve model performance.

Python's ability to seamlessly integrate with these machine learning models, along with its support for modern AI methodologies, ensures a streamlined and efficient development process. As AI continues to evolve, the role of Python as a foundational tool in this field is likely to grow even more significant, offering developers the speed, flexibility, and resources necessary to tackle increasingly complex challenges.

Keywords: Artificial intelligence, Python, machine learning, classification, electronic letters, Naive Bayes, Support Vector Machine (SVM).