DOI https://doi.org/10.32782/2078-0877-2025-25-3-12
УДК 004.42

І. А. Скрипник, канд. ф.-м. наук, доц. ORCID: 0000-0002-9175-2683
А. І. Безверхий, доцент ORCID: 0000-0002-0819-3690
Запорізький національний університет
e-mail: sia@zsea.edu.ua

ВЕКТОРИЗАЦІЯ ФОРМАЛЬНИХ ГРАМАТИК ДЛЯ ЇХ КЛАСТЕРИЗАЦІЇ ЗАСОБАМИ ML.NET

Анотація. Формальні граматики широко використовуються в компіляторах, методах обробки природних мов, аналізі коду та тестуванні програм. Застосування методів машинного навчання до граматик відкриває нові можливості для автоматизованого аналізу, класифікації, кластеризації та оптимізації мовних моделей. Для застосування методів кластерного аналізу потрібно «розмітити» набори даних, тобто перетворити правила граматики на числову форму. Застосовується автоматизований підхід для аналізу формальних граматик засобами ML.NET, зокрема метод векторизації TF-IDF, заснований на «зважуванні» граматичних символів і виявленні їх унікальності та впливу на контекст правил та мови, яку генерує формальна граматика. Результати векторизації граматик використано для їх кластеризації методом K-Means, доступним у ML.NET. Такий підхід забезпечить автоматизоване керування продукціями граматик та їх оптимізацію.

Ключові слова: формальна граматика, токен, векторизація, вхідна мова, граматичне правило, ML.NET, TF-IDF, кластеризація, K-Means.

Постановка проблеми. Традиційно формальні граматики аналізувалися вручну: фахівці переглядали правила або продукції, шукали дублікати, оптимізували правила та групували їх у схожі підмножини. Проте зі збільшенням розміру граматики, особливо у разі складних чи спеціалізованих мов програмування або складних моделей природної мови, ручний підхід стає затратним за часом і схильним до людських помилок. Із десятками чи сотнями правил складно виявити конструкції з однаковими ознаками [1].

Ускладнення традиційного аналізу пов’язано з такими факторами:

− Зростання розмірів граматики. Кілька десятків правил можуть породити сотні різних комбінацій нетерміналів і терміналів. Візуально й інтуїтивно відстежити ці залежності вкрай складно.

− Складність семантичних зв’язків. Декілька правил можуть формально відрізнятися, але семантично виконувати подібні функції. Ручний пошук таких «еквівалентних» шаблонів вимагає ґрунтовного розуміння граматики та тривалий за часом.

− Наявність аномальних чи неефективних конструкцій. У процесі активної роботи над граматикою можуть додатися нові правила, натомість старі дублюються або залишаються непотрібними. Без інструментів автоматизованого аналізу легко пропустити унікальні чи непродуктивні правила.

Через зазначені проблеми обробка граматики вручну стає не лише тривалою, а й ускладненою. Водночас автоматизація аналізу граматик відкриває можливість системного виявлення схожих правил, підрахунку частотності символів і побудови узагальнених кластерів, які за потреби в подальшому можна модифікувати.

Аналіз останніх досліджень. Сьогодні все більш популярними стають автоматизовані методи аналізу граматик із застосуванням алгоритмів машинного навчання для обробки природної мови, зокрема методи класифікації тексту з використанням N-грамних мовних моделей,

трансформерів для великих мовних моделей та автоматичного розпізнавання мовлення [2], методів кластеризації, у тому числі K-Means, методів векторизації з різними алгоритмами нормалізації для задач класифікації та машинного перекладу [3] та використання статистичного підходу для інтелектуального аналізу даних і прогнозування [4]. Окрім векторизації, граматика природно представляється як граф та застосовується ідея побудови абстрактного синтаксичного дерева (AST, Abstract Syntax Tree) [6]. У більшості робіт використовують засоби мов Python/R або бібліотеки для графових/нейронних моделей, зокрема PyTorch, TensorFlow. Бібліотека ML.NET не дуже часто використовується для дослідження проблем кластеризації об'єктів згідно з оглянутими публікаціями.

Формулювання мети статті (постановка завдання). Метою є дослідження методів векторизації формальних граматик для автоматизації їх аналізу методами кластеризації з подальшою оптимізацією продукцій.

Основна частина. Щоб застосувати методи кластерного аналізу, потрібно «розмітити» набори даних (або формалізувати), тобто перетворити правила граматики на числову форму. Одним зі способів такого перетворення є використання *векторного представлення*: кодування граматичних символів у формальних правилах за допомогою унікальних чисел або векторів.

У цьому контексті «дані» – це правила формальної граматики. При цьому «розмітка» означає перетворення правила з текстового рядка на числовий вектор. Саме так формується вхід для кластеризації: кожне правило стає точкою у багатовимірному просторі.

Згідно з класифікацією формальних граматик Н. Хомського [1], правила контекстно-вільної граматики G, яка породжує контекстно-вільну мову, мають вигляд: «<A> → α», де α – ланцюжок, можливо порожній, терміналів v∈V або нетерміналів w∈W, V – множина терміналів (вхідних слів), W – множина нетерміналів (допоміжних слів).

Алгоритм векторизації TF-IDF (Term Frequency-Inverse Document Frequency, зважування токенів у векторі) дає змогу оцінити важливість кожного токена (слова) в документі відносно всієї колекції документів [2; 3]. Кожне граматичне правило виду <A> → α розглядається як окремий документ, а кожен термінал або нетермінал у правій частині α – як токен.

TF-IDF об'єднує дві ідеї:

TF (*частота терміна*) – скільки разів термін з'являється в документі.

IDF (*обернена частота документів*) – як часто термін зустрічається в усіх документах (чим рідше – тим він важливіший).

У разі контекстно-вільної граматики зважуванню підлягають усі граматичні символи лівих та правих частин правил – термінали та нетермінали: обчислюються частота кожного граматичного символу в окремому правилі та частота того ж символу в усій граматиці.

Для обчислення числових векторів алгоритм TF-IDF [5] передбачає *попередню обробку* граматичних правил. Вона включає *нормалізацію* та *токенізацію*.

Нормалізація граматичних правил. Зазвичай у формальних граматиках нетермінальні символи як допоміжні записують у кутових дужках для наочної відмінності їх від термільналів із двох причин: 1. Для можливості іменування нетерміналів та терміналів словами (або символами) зі спільного словника. Наприклад, W = {<Alpha>, <A>, <Term>, …}, V = {A, Alpha, term, …}. 2. Для представлення процесу виведення речення вхідної мови в граматиці скінченною (або нескінченною) послідовністю проміжних ланцюжків граматичних символів: нетермінали фігурують у проміжних ланцюжках, поки виведення триває, а виведене в граматиці речення містить лише термінальні символи граматики $v_i \in V$. Для нормалізації правил виду «<A> → α» з них видаляють непотрібні розділові знаки, зокрема кутові дужки, стрілку та ліву частину, утворюючи з правої частини α один рядок символів. Часто для спрощення граматик нетермінали позначають великими літерами алфавіту, W = {A, B, C, …}, а термінали – малень-

кими, V = {a, b, c, …}, але це призводить до втрати інформативності елементів V, W та ускладнює аналіз граматик, що генерують неприродні мови, зокрема мови програмування. Таким чином, нормалізація граматики має ураховувати контекст правил та особливості вхідної мови.

Наступний етап – *токенізація* правил (розбивка на токени). Найпростіший спосіб – розбивка правої частини правила α на токени, що є односимвольними лексемами, тобто токени-символи. Це можливо, якщо всі термінали та нетермінали є символами. Щоб урахувати контекст та структуру правил, наприклад циклічність або рекурсивність, утворюють N-грами – послідовності з n токенів-символів, зокрема, біграми – пари послідовних токенів-символів, наприклад для α = «aBbAc» маємо множину біграм {«aB», «Bb», «bA», «Ac»}. Такий спосіб токенізації не підходить для граматик, у яких ∀ vi , wi , vi ϵ V, wi ϵ W, | vi | ≥1, | wi | ≥1. Для врахування багатосимвольних лексем правих частин правил, як то ідентифікатори, ключові слова, операції типу <=, ++ та ін., треба зробити власну токенізацію, тобто перетворити правила на такий вигляд, де багатосимвольні лексеми вже замінені унікальними токенами, що й утворять словник токенів. Таким чином, будь-яке входження кожного терміналу чи нетерміналу розглядається окремо. Важливо також урахувати порожнє правило A → ε (так зване ε-правило), ε – порожній ланцюжок. Для нього має бути введений у словник токен eps. У результаті токенізації має бути побудований *словник токенів* (Vocabulary), у який збирають усі унікальні токени з усіх правил граматики.

Після токенізації застосовується метод обчислення TF-IDF-векторів. Ознака TF: скільки разів кожен токен зустрічається в конкретному правилі. Ознака IDF: наскільки рідкісним є цей токен серед усіх правил граматики [5].

Формули TF-IDF з урахуванням уведених авторами позначень елементів формул, що відповідають контексту їх застосування для формальних граматик. Для токену t у правилі p із граматики G маємо:

1. TF (*t, p*) = частота токену t в правилі p.

2. DF (t, G) = $\log\left(\frac{N}{(1+DF(t))}\right)$, N – загальна кількість правил; DF(t) – кількість правил, у яких зустрічається токен t.

3. TF-IDF (t, p, G) = TF (t, p) × IDF (t, G).

TF-IDF вектори для продукцій КВ-граматики. Для словника токенів потужності n, кожне правило кодується як вектор із n компонент, де кожна компонента – це TF-IDF значення відповідного токену. У результаті кожне правило перетворюється на *вектор розмірності*, що дорівнює кількості унікальних токенів у всій граматиці, тобто потужності словника Vocabulary, |Vocabulary| = n. Кожна координата дорівнює добутку TF×IDF для відповідного токену. Таким чином, TF-IDF-вектор – це *вектор ознак*, де кожна координата відповідає унікальному токену граматики.

Побудовані TF-IDF-вектори можна використовувати як «розмічені дані» в задачах машинного навчання, зокрема кластеризації. При цьому виникає проблема представлення векторів великої розмірності. Для візуалізалізації TF-IDF-векторів застосовують методи зниження розмірності.

Метод PCA (Principal Component Analysis) дає змогу [5]:

- перетворити вектори великої розмірності у простір 2D або 3D;
- максимально зберегти *інформаційний уміст* початкових даних.

Це дасть можливість кластеризації граматик та візуалізації TF-IDF векторів у вигляді точок на площині.

Алгоритм PCA (2D): на вхід подається матриця X розміром (count_p × count_t), Xcount_p x count_t, рядками матриці є TF-IDF-вектори продукцій граматики, count_p – кількість правил, count_t = кількість ознак (унікальних токенів). Наступні кроки:

1. Центрування матриці. Від кожного стовпця (ознаки) віднімається середнє значення: $X_C = X - X_{сер}$.

2. Обчислення коваріаційної матриці:

$X_{COV} = (X_C^T * X_C)/(count\_p - 1)$

3. Знаходження власних векторів та власних значень матриці X_{COV} :

− Власні вектори – нові напрямки (головні компоненти).

− Власні значення – «вага» кожного напрямку.

4. Вибираємо дві головні компоненти. Вибираємо два власні вектори, які відповідають двом найбільшим власним значенням.

5. Проєкція в 2D. Матриця $W_{count\_t \times 2}$ (2 головні компоненти) має розмір (count_t × 2), і ми обчислюємо: $X_{2D} = X_C \times W$.

У результаті роботи алгоритму тримуємо координати для кожного правила у 2D-просторі.

У бібліотеці ML.NET [5] векторизація TF-IDF реалізується через набір трансформерів:

− TokenizeIntoWords / TokenizeIntoCharacters (розбивка рядка на токени);

− ProduceWordBags або ProduceNgrams із параметром WeightingCriteria.TfIdf (обчислення TF-IDF-векторів);

− NormalizeLpNorm (евклідова L_2-нормалізація векторів, щоб їх масштаб не впливав на кластерні алгоритми).

Після перетворення усіх продукцій граматики на TF-IDF-вектори можна застосувати кластеризацію. K-Means – один з алгоритмів, доступних у ML.NET. Він поділяє набір точок (векторів) на K кластерів, мінімізуючи внутрішньо кластерну відстань до центрів [4; 5].

Приклад. Вхідна контекстно-вільна мова являє собою оператори умовного переходу із вкладеними логічними виразами мовою C++. Контекстно-вільна граматика, що генерує задану мову, представлена у формі Бекуса – Наура [1]:

1) S0 -> if (S) { oper } E_EI

2) E_EI -> else CH | ε

3) CH -> S0 | { oper }

4) S -> (S) C | ! idp AO | idp C

5) AO -> || S | && S | ε

6) C -> != S1 | == S1 | > F1 | < F1 | AO | ε

7) F1 -> idp AO | = idp AO

8) S1 -> (S) AO | idp AO | ! idp AO

Словник токенів Vocabulary = { S0, ->, if, (, S,), {, oper, }, E_EI, else, CH, ε, C, !, idp, AO, ||, &&, !=, S1, ==, >, F1, <, = }, | Vocabulary | = 26. Для такого словника TF-IDF-вектори будуть мати довжину 26. TF-IDF-вектор для першого правила з лівою частиною – аксіомою граматики – має вигляд:

(2,398; 0,00; 3,091; 1,992; 1,145; 1,992; 2,398; 2,398; 2,398; 1,992; 0,000; 0,00; 0,00; 0,00; 0,00; 0,00; 0,00; 0,00; 0,00; 0,00; 0,00; 0,00; 0,00; 0,00; 0,00; 0,00).

Видно, що менше половини ознак (координат) вектору ненульові, тобто вектор зберігає лише частину якісної інформації про токени граматичної продукції. Перетворення вектору ознак великої довжини за допомогою алгоритму PCA до точки площини дає такі результати:

PCA to 2D Coordinates: S0 -> if (S) { oper } E_EI => (-5,905, -0,175).

Обчислені для всіх граматичних правил 2D-вектори використовуються для кластеризації методом K-Means. Результати групування правил зображено на рис. 1.

Кожному правилу відповідає точка площини: PCA1 задає абсцису, PCA2 задає ординату, точки одного кольору належать окремому кластеру. Зазначимо, що в окремих кластерах згрупувалися правила, схожі за будовою, тобто вмістом та послідовністю граматичних символів –

токенів. Оскільки векторизація правил виявила їхні якісні ознаки, представлені токенами, то розбиття на кластери фактично виконується за генеративною функцією граматичних правил: правила з одного кластеру генерують аналогічні речення вхідної (формальної) мови.

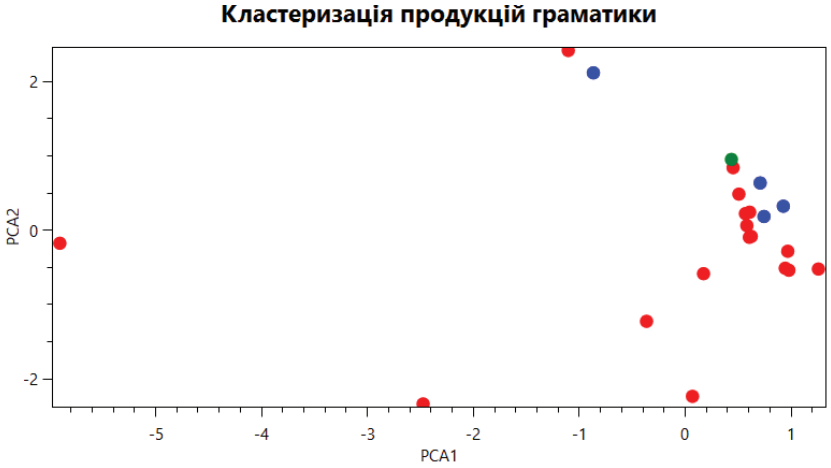

Рис. 1. Результати кластеризації правил контекстно-вільної граматики за алгоритмом K-середніх

Аналіз кластерів дає змогу знаходити групи схожих правил, виявляти аномальні чи рідкісні правила, оптимізувати граматики. Автоматизований підхід дає змогу працювати з великими граматиками швидко та точно, виявляти приховані семантичні закономірності, забезпечує масштабованість. Результати векторизації граматик можна використовувати в інших алгоритмах машинного навчання, наприклад для класифікації граматик за стилем, порівняння різних версій граматики чи виявлення схованих шаблонів.

Висновки. Запропоновано автоматизований підхід для аналізу формальних граматик. Проаналізовано засоби бібліотеки ML.NET для векторизації формальних граматик та подальшої кластеризація. Установлено, що векторизація TF-IDF та K-Means в ML.NET надають потужний інструмент для автоматизованого керування та оптимізації граматичних правил. Виявлено подальші перспективи автоматизованого аналізу формальних граматик.

Список використаних джерел

1. Aho A., Sethi R., Ullman J. Compilers: principles, technicques and tools. Addison-Wesley: Reading, MA, 1986.

2. Jurafsky D., Martin J.H. Speech and Language Processing. 2025. URL: https://web.stanford.edu/~jurafsky/slp3/ (дата звернення: 10.05.2025).

3. Manning C.D., Raghavan P., Schütze H. Introduction to Information Retrieval. Cambridge University Press. 2008. URL: https://nlp.stanford.edu/IR-book/pdf/irbookonlinereading.pdf (дата звернення: 10.05.2025).

4. Hastie T., Tibshirani R., Friedman J. The Elements of Statistical Learning. URL: https://www.sas.upenn.edu/~fdiebold/NoHesitations/BookAdvanced.pdf (дата звернення: 02.05.2025).

5. Офіційна документація ML.NET. URL: https://learn.microsoft.com/en-us/dotnet/api/microsoft.ml.transforms?view=ml-dotnet-preview (дата звернення: 07.05.2025).

6. Alon U., Zilberstein M., Levy O., Yahav E. code2vec: Learning Distributed Representations of Code (POPL 2019). URL: https://arxiv.org/abs/1803.09473?utm_source=chatgpt.com (дата звернення: 15.08.2025).

Стаття надійшла до редакції 26.09.2025
Стаття прийнята 14.10.2025
Статтю опубліковано 25.11.2025

I. Skrypnyk, A. Bezverkhyi
Zaporizhzhia National University

VECTORIZATION OF FORMAL GRAMMARS FOR THEIR CLUSTERING USING ML.NET

Summary

Formal grammars are widely used in compilers, natural language processing methods, code analysis, and software testing. The application of machine learning methods to grammars opens new opportunities for automated analysis, classification, clustering, and optimization of language models. To apply clustering methods, it is necessary to «label» datasets, i.e., to transform grammar rules into a numerical form. An automated approach for analyzing formal grammars using ML.NET is applied. In particular, the TF-IDF vectorization method is used, based on «weighting» grammatical symbols and identifying their uniqueness and influence on the context of rules and the language generated by the formal grammar. The results of grammar vectorization are used for their clustering with the K-Means method available in ML.NET. This approach provides automated management of grammar productions and their optimization.

The results of vectorization and subsequent clustering show that rules similar in structure that is, in the content and sequence of grammatical symbols (tokens) were grouped together within specific clusters. Since the vectorization of rules revealed their qualitative features represented by tokens, the clustering is essentially performed according to the generative function of the grammatical rules: rules within the same cluster generate similar sentences of the input (formal) language.

Conclusions. An automated approach for analyzing formal grammars has been proposed. ML.NET tools for vectorization of formal grammars and subsequent clustering were analyzed. It was established that TF-IDF vectorization and K-Means in ML.NET provide a powerful tool for automated management and optimization of grammatical rules. Further prospects for automated analysis of formal grammars were identified.

Keywords: formal grammar, token, vectorization, input language, grammar rule, ML.NET, TF-IDF, clustering, K-Means.