002.2 Векторна модель документа

У комп’ютерній лінгвістиці існує базовий принцип: будь-яке мовне явище, аби стати об’єктом формальної обробки, спершу необхідно подати у числовому вигляді. Однією з найдавніших і водночас найпоширеніших технік такої репрезентації є модель «мішок слів» – Bag-of-Words, далі BoW. Її ідея проста, але продуктивна: кожен документ розглядається як невпорядкована множина лексем, порядок їх появи, граматика та синтаксис ігноруються, а увага зосереджується винятково на тому, чи присутнє певне слово і скільки разів воно входить до тексту. На перший погляд це знеособлює мовний матеріал, позбавляючи його багатої структурної інформації. Проте саме така редукція дає змогу перетворити текст на вектор, а далі скористатися перевіреним арсеналом лінійної алгебри та статистики – від класифікації електронних листів до пошуку дотичних наукових статей.

Щоби застосувати BoW, дослідникові передусім треба ухвалити рішення стосовно корпусу, тобто сукупності документів. У процесі побудови «словникового запасу» усі тексти поєднують, а з отриманої суцільної послідовності вилучають непотрібні символи, розділові знаки та надмірні пропуски. Текст переводять у нижній регістр, аби форми *Machine* та *machine* не створювали зайвих вимірів. Далі задіяна токенізація, що розбиває суцільний рядок на окремі лексеми; до прикладу, вислів *I love machine learning* після токенізації стане послідовністю клаптиків *I, love, machine, learning*. Опційно відфільтровують зупинні слова, характерні для кожної мови службові частки, котрі не несуть змістовного навантаження і лише збільшують розмірність векторного простору.

Унаслідок описаних кроків формується словник – впорядкований список усіх унікальних слів, що зосталися після попередньої обробки. Припустімо, що колекцію утворюють три невеличкі уривки: *Machine learning is fun*, *Machine learning is important*, а також *Fun is important*. Тоді остаточний словник міститиме лексеми *machine, learning, is, fun, important*. Кожне слово відповідає певній координаті у векторі, а документи стають точками у п’ятивимірному просторі. Перша фраза буде вектором ⟨1, 1, 1, 1, 0⟩, друга – ⟨1, 1, 1, 0, 1⟩, а третя – ⟨0, 0, 1, 1, 1⟩. Така репрезентація наочно демонструє, що перші два уривки подібні, оскільки відрізняються лише в останньому компоненті.

Принцип «все, що є, має бути враховане» поширюється й на машинне навчання. Маючи BoW-вектори, легко обчислити косинусову подібність між документами. Косинус кута θ між векторами d_1 та d_2 дорівнює скалярному добутку, поділеному на добуток їх норм: $\cos\theta = (d_1 \cdot d_2)/(\|d_1\|\,\|d_2\|)$. Значення, наближені до одиниці, вказують на схожість тематики, тоді як нульовий результат свідчить про повну розбіжність лексем. У нашому прикладі перші два вислови мають косинус 0,8, що явно більше, ніж 0,5 у пари першого та третього.

Частотний підхід має суттєві переваги. По-перше, він елементарно реалізується програмно. У Python достатньо скористатися класом *CountVectorizer* з пакета *scikit-learn*. Ініціалізація об’єкта без параметрів дає інструмент, що автоматично виконує

токенізацію й трансформує корпус в розріджену матрицю. Кожен рядок цієї матриці відповідає документу, кожен стовпчик – слову. Приклад коду можна викласти стисло:

```
from sklearn.feature_extraction.text import CountVectorizer
docs = ["I love machine learning", "Machine learning is amazing"]
cv = CountVectorizer(stop_words='english')
X = cv.fit_transform(docs)
print(cv.get_feature_names_out())
print(X.toarray())
```

Вивід покаже словник *amazing, learning, love, machine* і дві векторні строки: ⟨0, 1, 1, 1⟩ та ⟨1, 1, 0, 1⟩. Наочно видно, що слово *love* відсутнє у другому реченні, а *amazing* не зустрічається в першому.

По-друге, BoW чудово функціонує на коротких фрагментах, де граматика не має вирішального значення. Система виявлення спаму використовує саме цей принцип: підраховуючи, як часто трапляються лексеми *free*, *discount* чи *subscribe*, вона визначає, чи варто класифікувати лист як небажаний.

Та попри зручність, модель «мішок слів» позбавлена чутливості до порядку і контексту. Словосполучення *not good* та *good* у векторному представленні відрізняються лише одним елементом, і якщо обидва слова часто трапляються порізно, алгоритм ризикує ототожнити семантично протилежні твердження. Друга слабка ланка – висока розмірність: розгорнутий новинний архів може містити сотні тисяч унікальних лексем, і кожна породжує окрему координату. Матриця стає надто розрідженою, більшість комірок містить нулі, а обчислення – обтяжливими.

З огляду на згадані недоліки розробники вдосконалили BoW шляхом запровадження вагової схеми TF–IDF. Насамперед потрібно пояснити компоненти. Частотність терміна TF(t,d) – це кількість входжень слова t у документ d. Щоб прибрати упередження щодо довших текстів, частоту нормують, поділяючи на загальну кількість токенів у документі. Уявімо, що в рецензії на книжку двісті слів, і слово *masterpiece* зустрічається чотири рази. Нормована TF дорівнюватиме 4 / 200 = 0,02.

Друга складова – обернена частотність документів IDF(t). Формула log(N/df(t)) враховує, у скількох текстах з N-елементного корпусу з’являється лексема t. Якщо слово трапляється всюди, log-чисельник прямує до нуля, відповідно, IDF зменшується і вага стає мізерною. Наприклад, нехай у базі даних десять тисяч статей, а термін *data* є у сотні з них: IDF дорівнює log(10000/100)≈2. Якщо ж розглядати вузькоспеціальний вираз *spectral clustering*, що трапився лише в трьох статтях, IDF зросте до log(10000/3)≈8,1, наголошуючи на його унікальності.

Поєднання нормованої TF і IDF народжує вагу TF–IDF = TF·IDF. Це число описує, наскільки слово характерне для конкретного документа і водночас нетипове для інших. Саме так пошуковики визначають релевантність ключових запитів: терміни, які часто з’являються в одному тексті, але рідкі стають сигналом тематичної приналежності. Для прагматичного прикладу розглянемо код.

```
import math, re, collections
docs = ["This is a sample document.",
        "We study document frequency and inverse document
frequency.",
        "This document is about machine learning and data
science.",
        "Another document examines natural language processing.",
        "Machine learning is a branch of data science."]

def tf(term, doc):
    tokens = re.findall(r"\w+", doc.lower())
    return tokens.count(term) / len(tokens)

def idf(term, corpus):
    N = len(corpus)
    df = sum(1 for d in corpus if term in d.lower())
    return 0 if df == 0 else math.log(N / df)

term = "document"
print("TF first doc:", tf(term, docs[0]))
print("IDF corpus:", idf(term, docs))
print("TF-IDF first doc:", tf(term, docs[0]) * idf(term, docs))
```

Результат покаже, що у першому реченні TF терміна *document* становить приблизно 0,111, IDF – 0,5108, а їх добуток – 0,056, через що слово не надто вплине на оцінку подібності.

Поки ми оперували поодинокими векторами, та на практиці доводиться створювати термін-документну матрицю. Рядки відповідають словам, стовпці – текстам. Кожна клітинка містить або сирий підрахунок, або TF–IDF коефіцієнт. Виходить гігантський, але переважно порожній масив, що зберігають у стиснутому форматі CSR. Таку структуру вже можна подавати на вхід класифікаторам: наприклад, SVM відмінно працює із розрідженими даними та здатен відокремити статті з теми «Комп'ютерні мережі» від «Теорії графів», спираючись на розподіл лексем.

Якщо BoW та TF–IDF розглянути під кутом геометрії, легко усвідомити, що ми маємо справу з класичною Векторною моделлю простору документів – Vector Space Model, скорочено VSM. У цій парадигмі кожен текст ототожнюють із точкою у n-вимірному евклідовому просторі, n дорівнює кількості термінів у словнику. Вектор $d_i = \langle w_{11} \dots w_{□1} \rangle$, де $w_{i□}$ – вага i-го слова в j-му документі. Завдяки такій абстракції можна застосувати косинусову метрику, евклідову відстань чи коефіцієнт Жаккара для оцінки близькості будь-якої пари точок. У системі пошуку користувацький запит трансформують так само, як і документи. Далі обчислюють косинус між вектором запиту q і векторами текстів, сортують результати за спаданням значення cos θ, подаючи на верхні позиції ті, що стоять ближче до q.

Попри елегантність, VSM успадковує обмеження BoW. Порядок слів залишається поза увагою, тому змінена позиція заперечення знищує коректне тлумачення. Модель не

знає, що *автомобіль* і *машина* позначають те саме, бо кожне слово – окрема координата. Хоча у науковій літературі обговорюють варіанти розширення словника синонімами або використання тезаурусів WordNet, проблема полі- та омонімії остаточно не вирішена.

Надмірна розмірність також проблематична: у великих базах даних кількість унікальних токенів сягає сотень тисяч. Методи зменшення, такі як латентне семантичне індексування чи стохастичне відсікання ознак, частково розв'язують задачу, але водночас ускладнюють інтерпретацію результатів. Сучасні нейронні підходи, скажімо Word2Vec, пропонують компактні вбудовування, що зберігають семантику у кількасотвимірному просторі, проте тоді втрачається прямий зв'язок із вихідними лексемами.

Утім, практичний внесок VSM залишається колосальним. У пошукових системах інформація індексується саме у вигляді термін-документної матриці, з можливістю швидкого ранжування результатів під запит. У системах рекомендацій список статей для прочитання формують, беручи найперші k елементів із найбільшим косинусом до історії переглядів користувача. Кластеризація новинного потоку за допомогою k-means або DBSCAN спирається на ті ж вектори, дозволяючи відстежити появу нових тем у режимі реального часу.

Щоби підсумувати, варто ще раз окреслити логіку руху від тексту до числа. Спершу корпус проходить через каскад очищення: видалення пунктуації, перехід у нижній регістр, фільтрація зупинних слів, лематизація. Далі формується словник, для кожного документа рахують частоти або ваги TF–IDF. Отримані вектори лягають у термін-документну матрицю, що стає підмурком для вимірювання подібності, класифікації, тематичного моделювання і багато чого іншого. Попри здавалося б обмежений погляд – «слова без порядку» – BoW і VSM уже кілька десятиліть залишаються надійними інструментами, зручними у випадках, де потрібна швидкість, масштабованість і прозорість, а не глибоке мовознавче розуміння. Такі методи й сьогодні активно застосовують у галузях, де головною цінністю є не суто лінгвістична точність, а сумарна ефективність системи: від аналізу великого обсягу юридичних документів до ранньої діагностики кібератак за журналами подій.

Водночас сучасні тренди скеровують дослідницьку спільноту до більш витончених підходів. Трансформерні архітектури BERT чи GPT здатні навчатися контексту та семантичних зв'язків, яким BoW байдужий. Проте й найскладніші моделі на етапі підготовки даних усе ще користуються базовими прийомами токенізації й нормалізації, успадкованими від BoW. Тож у певному сенсі «мішок слів» залишився фундаментом, на якому вибудовують сучасні хмарочоси нейромережевих технологій.

Література для поглибленого вивчення теми

Основи векторної моделі документа (Bag-of-Words, TF-IDF, Vector Space Model):

1. SALTON, G., WONG, A., & YANG, C. S. (1975). A Vector Space Model for Automatic Indexing. *Communications of the ACM*, 18(11), 613–620. URL: https://dl.acm.org/doi/10.1145/361219.361220, DOI: 10.1145/361219.361220
2. MANNING, C. D., RAGHAVAN, P., & SCHÜTZE, H. (2008). *Introduction to Information Retrieval*. Cambridge University Press. URL: https://nlp.stanford.edu/IR-book/html/htmledition/
3. JURAFSKY, D. & MARTIN, J. H. (2009). *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition* (2nd ed.). Prentice Hall.

Попередня обробка тексту (токенізація, нормалізація, зупинні слова, стемінг, лематизація):

4. BIRD, S., KLEIN, E., & LOPER, E. (2009). *Natural Language Processing with Python – Analysing Text with the Natural Language Toolkit*. O'Reilly Media. URL: https://www.nltk.org/book/
5. PORTER, M. F. (1980). An Algorithm for Suffix Stripping. *Program: electronic library and information systems*, 14(3), 130–137. URL: https://doi.org/10.1108/eb047160, DOI: 10.1108/eb047160

Практичні реалізації та бібліотеки (scikit-learn, CountVectorizer, TfidfVectorizer):

6. PEDREGOSA, F., VAROQUAUX, G., GRAMFORT, A., MICHEL, V., THIRION, B., GRISEL, O., ... & DUCHESNAY, E. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12, 2825–2830. URL: https://www.jmlr.org/papers/volume12/pedregosa11a/pedregosa11a.pdf
7. *Scikit-learn User Guide: Text feature extraction*. (Актуальна версія). URL: https://scikit-learn.org/stable/modules/feature_extraction.html#text-feature-extraction

Вимірювання подібності (косинусна подібність, евклідова відстань):

8. HAN, J., KAMBER, M., & PEI, J. (2011). *Data Mining: Concepts and Techniques* (3rd ed.). Morgan Kaufmann.

Обмеження BoW та VSM, а також сучасні підходи (Word Embeddings, Transformers):

9. MIKOLOV, T., CHEN, K., CORRADO, G., & DEAN, J. (2013). Efficient Estimation of Word Representations in Vector Space. *arXiv preprint arXiv:1301.3781*. URL: https://arxiv.org/abs/1301.3781, DOI: 10.48550/arXiv.1301.3781
10. DEVLIN, J., CHANG, M.-W., LEE, K., & TOUTANOVA, K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. У *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)* (pp. 4171–4186). Association for Computational Linguistics. URL: https://aclanthology.org/N19-1423/, DOI: 10.18653/v1/N19-1423