2025-02-24

Векторизация, кодирование и языковые свертки (word embedding)

Чтобы слабоструктурированные (и часто гетерогенные) данные можно было сравнивать между собой, обычно применяется некоторый механизм приведения их в вид точек на условной плоскости.

При этом для эффективного различения слабоструктурированных данных между собой необходимо использовать пространство большой размерности ($n > 100$). Это вынужденная мера, т.к. кодирования данных малым число параметров (измерений) приводит к тому, что разные с точки зрения предметной области / семантики данные группируются вместе или оказываются полностью неразличимы.

Кодирование осуществляется по некоторому правилу, которое сопоставляет каждому измерению какое-либо числовое значение. Это позволяет получать одинаковые по длине векторы (коды) любого элемента слабоструктурированных данных.

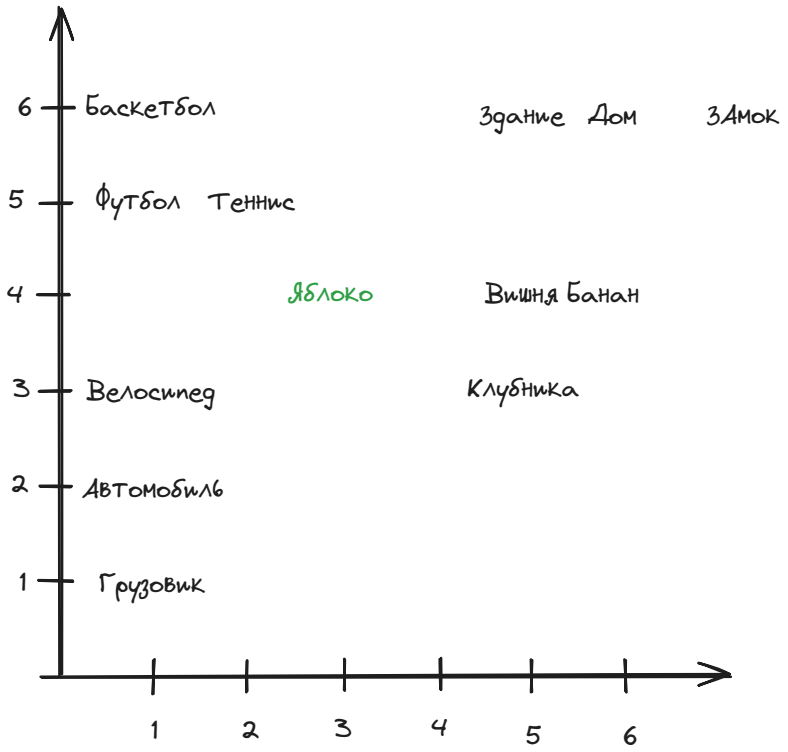
Правила могут быть различны и выбираются на этапе кодирования (векторизации).

Рассмотрим на примере.

Дан список слов: `Банан, Баскетбол, Велосипед, Здание, Автомобиль, ЗАмок, Вишня, Дом, Футбол, Клубника, Теннис, Грузовик`

Предположим, что по выбранному нами правилу «Банан» -> `[6, 4]`

1. Какие тогда могут быть координаты у других известных слов?
2. Куда можно будет разместить «Яблоко»?



Такой пример показывает недостаточность двух измерений для эффективного кодирования большого объема слабоструктурированных данных.

С увеличением числа элементов (в данном случае — увеличение вокабуляра, т.е. словаря уникальных слов) все сложнее отделить даже разные по семантике элементы друг от друга с достаточной для решения прикладных задач точностью — расстояние между ними будет на уровне погрешности, что не позволяет применять такое кодирование в прикладных задачах.

Поэтому **всегда** кодируют многомерными векторами.

Бинарное кодирование (one-hot encoding)

Одним из наиболее простых подходов к кодированию является бинарное кодирование.

Пример:

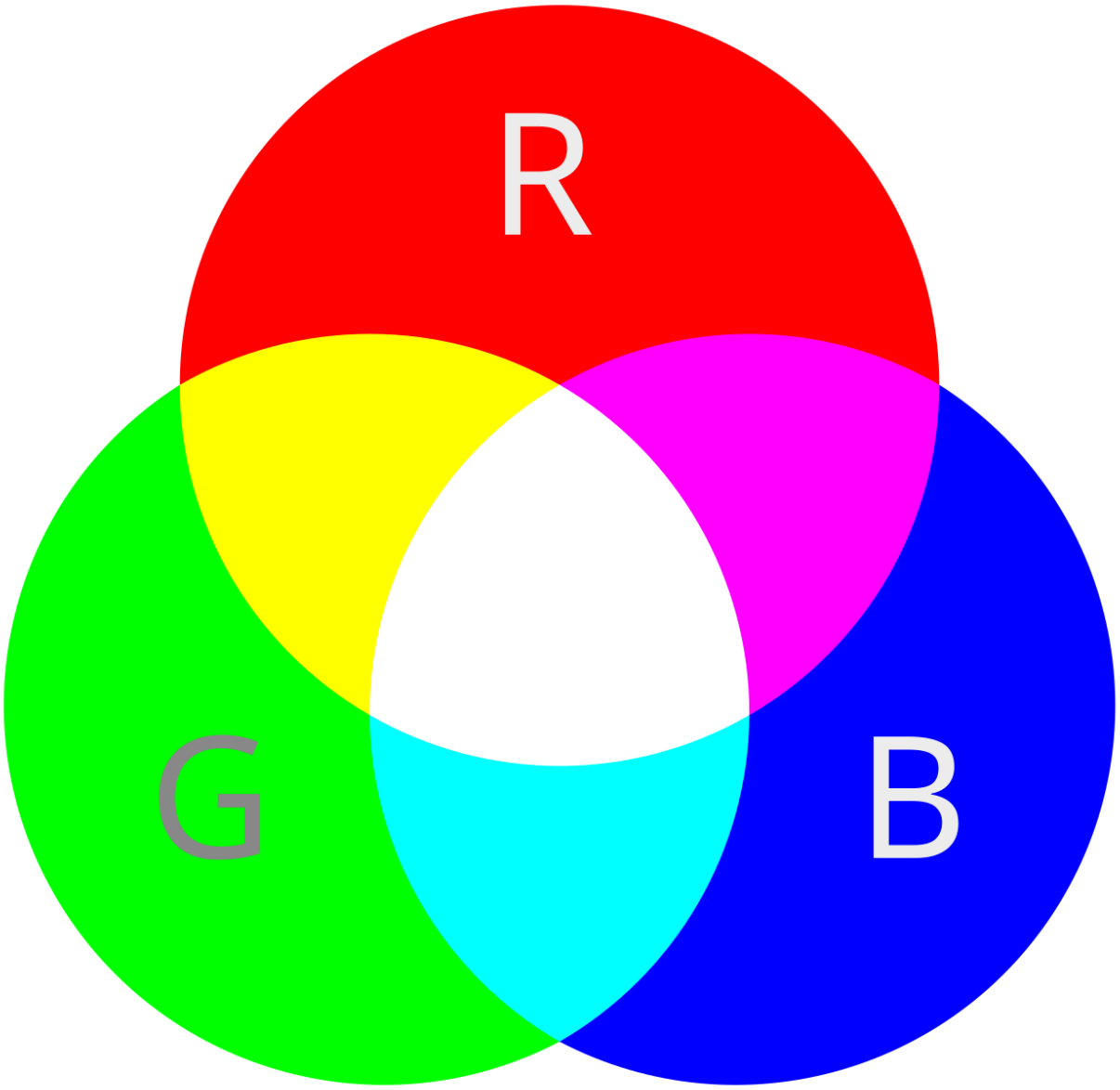
Есть 3 цвета: красный, зеленый и синий. Их всего 3, каждый цвет — 1 из 3 возможных. Можно сказать, что при расположении их в виде матрицы 3×3 получится диагональ, где «красный» будет соответствовать красному, зеленый — зеленому и синий — синему:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

```
[
    [1, 0, 0], # красный
    [0, 1, 0], # зеленый
    [0, 0, 1]  # синий
]
```

Суть бинарного кодирования заключается в том, что производится зануление всех измерений, кроме значимых. Значимые измерения приводятся всегда к единице. Отсюда и название — каждый элемент кодируется бинарным кодом, длина которого совпадает с число выбранных измерений.

Признаки в бинарном кодировании могут выбираться как извне (экспертно, статистически или иными методами), так и изнутри (как в случае с цветами, где каждое измерение — один из базовых цветов, которые кодируют сами себя; этот подход прост, но может приводить к разреженности — данные оказываются слишком далеко друг от друга, что снижает качество решений прикладных задач).




```
[
    [1, 0, 0], # красный
    [0, 1, 0], # зеленый
    [0, 0, 1], # синий
    [1, 1, 0], # желтый
    [0, 1, 1], # бирюзовый
    [1, 0, 1], # фиолетовый
    [0, 0, 0], # черный
    [1, 1, 1]  # белый
]
```

Бинарное кодирование близко по своей сути к тому, как кодируются объекты в памяти компьютера, что в некоторых классах задач позволяет не придумывать

новые правила, а брать правила, уже используемые для их представления в памяти компьютера (16-битный цвет кодируется 16 признаками и т.п.)

Бинарное кодирование категориальных данных

Бинарное кодирование удобно для категориальных данных (т.е. данных, которые легко представляются в виде таблицы, где количество уникальных значений в каждом столбце невелико)

Допустим, надо закодировать сотрудников по трем категориям: пол, образование, отдел.

Есть такая таблица (может быть и JSON, и XML и т.п.):

№ сотрудника	Пол	Образование	Отдел
112	М	Среднее	А
113	М	Высшее	А
114	Ж	Среднеспец	А
136	Ж	Высшее	Б
142	Ж	Высшее	Б
199	М	Среднее	В
201	Ж	Среднеспец	В

1. Смотрим, сколько уникальных значений в каждой категории:
    1. Пол — 2
    2. Образование — 3
    3. Отдел — 3
2. Для каждой пары категории и ее значения мы создаем отдельное измерение в векторе, что можно представить как новую таблицу (см. ниже)
3. Далее для каждого исходного элемента заполняем новую таблицу, но только 1 и 0

№ сотрудника	Пол_М	Пол_Ж	Обр_Выс	Обр_ССпец	Обр_Сред	Отдел_А	Отд
112	1	0	0	0	1	1	0
113	1	0	1	0	0	1	0
114	0	1	0	1	0	1	0
136	0	1	1	0	0	0	1
142	0	1	1	0	0	0	1
199	1	0	0	0	1	0	0
201	0	1	0	1	0	0	0

112 — `[1, 0, 0, 0, 1, 1, 0, 0]`

201 — `[0, 1, 0, 1, 0, 0, 0, 1]`

Готовая реализация one-hot encoding есть в Scikit-Learn: OneHotEncoder — scikit-learn 1.6.1 documentation

Лабораторная работа №4: Векторное представление текста и его кластерный анализ

1. Дана коллекция текстовых документов `2021_SPORT`
2. Необходимо провести их кластерный анализ методом спектральной кластеризации и визуализировать кластеры
   1. Вам понадобится NLTK, PyMorphy 3 для обработки ЕЯ, Scikit-Learn, SciPy, NumPy, опционально pandas для кластеризации, matplotlib и seaborn для визуализации
   2. Для этого каждый текст необходимо предварительно обработать: токенизировать (изучите `nltk.tokenize`), удалить стоп-слова (`nltk.corpus`, нормализовать (лемматизировать) (`pymorphy3`).
   3. Затем из обработанных текстов нужно составить корпус — коллекцию документов в нормальной форме
   4. Корпус передается в качестве аргумента в векторизатор (см. `CountVectorizer` в Scikit-Learn, пока считайте только униграммы, максимум признаков — 100-150, можно экспериментировать)
   5. Затем нужно построить матрицу расстояний (`squareform`, `pdist` из SciPy), для текстов чаще всего берут косинусное расстояние (`cosine`) и

визуализировать ее в виде тепловой карты (Seaborn вам в помощь)

6. Далее кластеризуем, проверяем коэффициент силуэта — может оказаться, что лучше покажет себя такая настройка: `SpectralClustering(affinity='precomputed', random_state=0`, в этом случае нужно матрицу расстояний перевести в матрицу сходства через формулу: `similarity_matrix = np.exp(- matrix ** 2 / (2. * delta ** 2))`, где `delta` — ширина Гауссова ядра, рекомендую начать где-то с 4.0

7. Визуализируем полученные кластеры