



КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ
ТАРАСА ШЕВЧЕНКА

ЗВІТ ДО ЛАБОРАТОРНОЇ РОБОТИ №2

**Контекстне автодоповнення вводу тексту
за допомогою N-грам**

Пядика Любомира, МІ-4

24 листопада 2018 р.

1 Постановка задачі

Необхідно розв'язати задачу знаходження найдоцільніших контекстних рекомендацій при вводі довільного тексту. При реалізації використати N-грами, а саме - біграми, які власне і ранжируються у порядку спадання доцільності при вводі нового слова користувачем. Для знаходження ранжувань використати два методи: метод найбільшої правдоподібності та метод згладжування Лапласа. А також, для демонстрації розробити дружній інтерфейс системи.

2 Теоретичні відомості

N-грама [3] — послідовність з n елементів. З семантичної точки зору, це може бути послідовність звуків, складів, слів або букв. На практиці частіше зустрічається N-грами як ряд слів, стійкі словосполучення називають колокацією. Послідовність з двох послідовних елементів часто називають біграм, послідовність з трьох елементів називається триграма. Не менш чотирьох і вище елементів позначаються як N-грами, N замінюється на кількість послідовних елементів.

Метод максимальної правдоподібності [2] (також метод найбільшої вірогідності) у математичній статистиці — це метод оцінювання невідомого параметра шляхом максимізації функції правдоподібності. Він ґрунтується на припущенні про те, що вся інформація про статистичну вибірку міститься у цій функції. Метод максимальної правдоподібності був проаналізований, рекомендований і значно популяризований Р. Фішером між 1912 і 1922 роками (хоча раніше він використовувався Гаусом, Лапласом і іншими). Оцінка максимальної правдоподібності є популярним статистичним методом, який використовується для створення статистичної моделі на основі даних, і забезпечення оцінки параметрів моделі.

Метод максимальної правдоподібності відповідає багатьом відомим методам оцінки в області статистики. Наприклад, припустимо, що ви зацікавлені зростом мешканців України. Припустимо, у вас дані стосовно зросту деякої кількості людей, а не всього населення. Крім того передбачається, що зріст є нормально розподіленою величиною з невідомою дисперсією і середнім значенням. Вибіркові середнє значення і дисперсія зросту є максимально правдоподібними до середнього значення і дисперсії всього населення.

Для фіксованого набору даних і базової імовірнісної моделі, використовуючи метод максимальної правдоподібності, ми набудемо значень параметрів моделі, які роблять дані “ближчими” до реальних. Оцінка максимальної правдоподібності дає унікальний і простий спосіб визначити рішення у разі нормального розподілу.

При N дослідженнях спостереження $(x) = \langle x_1, x_2, \dots, x_d \rangle$ методом максимальної правдоподібності матиме оцінку: $\hat{\theta}_i = \frac{x_i}{N}$.

Адитивне згладжування [1], також називається *згладжуванням Лапласа*, або згладжування “*Лідстоун*” - це техніка, яка використовується для вирівнювання категоріальних даних. Дано спостереження $(x) = \langle x_1, x_2, \dots, x_d \rangle$ з N дослідженнями, “згладжена” версія даних дає оцінку: $\hat{\theta}_i = \frac{x_i + \alpha}{N + \alpha d}$ ($i = 1, \dots, d$), де $\alpha > 0$ - це параметр згладжування. $\alpha = 0$ відповідає відсутності згладжування.

3 Програмна реалізація

Лабораторну роботу було написано на мові програмування Python. Програма складається із 3 файлів (autocomplete.py - головний файл, який виконує навчання і взаємодіє з користувачем; count_dict.py - реалізовує клас, який читає датасет, і будує необхідні структури із нього, як словники уніграм та біграм тощо; probabilities.py - реалізовує метод максимальної

правдоподібності та згладжування Лапласа для знаходження рекомендацій для кожного біграма).

Для тестування було використано різні навчальні тексти, у тому числі “Bible” і “Alice in the Wonderland”, які доступні у публічному доступі в інтернеті.

3.1 Приклад виконання програми

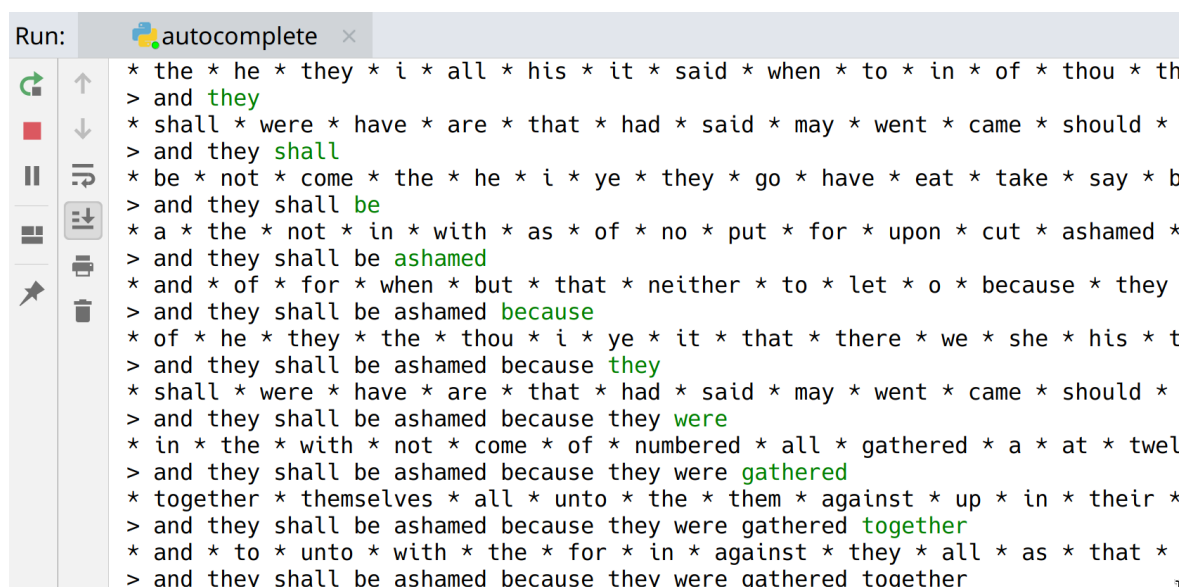


Рис. 1: Запущений autocomplete.py файл у PyCharm

3.2 Лістинг програми

autocomplete.py

```
import sys

from bigram_autocomplete.count_dict import CountDict
from bigram_autocomplete.probabilities import ProbabilityDict

def main():
    train_path = "../data/bible.txt"
    smoothing = "laplace"

    # читаємо навчальний текст і
    # обчислюємо для нього уніграми, біграми
    counts = CountDict(train_path)
    counts.populate()

    # обчислюємо достовірності кожної біграми
    # обраним методом
    probs = ProbabilityDict(counts)

    if smoothing == "mle":
```



```

# зчитуємо навчальний файл порядково
with open(self.filename) as f:
    for line in f:
        line = line.lower()
        sentences = line.split('.')

        # виділяємо окремо речення, і структурні елементи речень,
        # щоб уникнути помилкових рекомендацій
        for sentence in sentences:
            # переводимо рядок до нижнього регістру
            # і вилучаємо символи за допомогою попередньої регулярки
            words = to_remove.sub("", sentence)

            # розділяємо рядок на токени (слова)
            tokens = words.split()

            # заповнюємо словник уніграм,
            # підраховуючи кількі кожної
            for word in tokens:
                self.unigrams[word] += 1

            # заповнюємо словник біграм,
            # підраховуючи кількі кожної
            for i in range(len(tokens) - 1):
                bigram = (tokens[i], tokens[i + 1])
                self.bigrams[bigram] += 1

# к-ть унікальних уніграм та біграм
self.unique_unigrams = len(self.unigrams.keys())
self.unique_bigrams = len(self.bigrams.keys())

# заповнюємо словник відповідностей між словом
# там можливими доповненнями згідно з навчальним файлом
self.bigram_profile = defaultdict(set)

for key in self.bigrams:
    w1, w2 = key
    self.bigram_profile[w1].add(key)

```

probabilities.py

```

from collections import defaultdict

```

```

# допоміжний клас, який у разі відсутності ключа
# заповнює певним значенням, яке видає фабрика
class CustomDict(dict):
    def __init__(self, factory):
        self.factory = factory

    def __missing__(self, key):

```

```

        self[key] = self.factory(key)
        return self[key]

# допоміжний клас для знаходження достовірностей біграм
class ProbabilityDict(object):
    def __init__(self, count_dict):
        self.counts = count_dict

        # знаходиться кількість унікальних слів та загальна кількість вжитих слів
        # та виводиться на екран
        self.vocab_size_total = sum(self.counts.unigrams.values())
        self.vocab_size_unique = len(self.counts.unigrams.keys())
        print("Vocab size: total={}, unique={}".format(self.vocab_size_total, self.vocab_size_unique))

    # Maximum Likelihood Estimate
    # методом максимальної достовірності знаходяться достовірності біграм
    def bigram_MLE(self):
        mle = defaultdict(float)
        for key, value in self.counts.bigrams.items():
            w1, w2 = key
            mle[key] = value / self.counts.unigrams[w1]

        return mle

    # Additive or Laplace smoothing
    # методом Лапласового згладжування знаходяться достовірності біграм
    def bigram_laplace(self):
        laplace_dict = defaultdict(float)

        # коефіцієнт згладжування
        alpha = 1

        for key, value in self.counts.bigrams.items():
            w1, w2 = key
            laplace_dict[key] = (value + alpha) / (self.counts.unigrams[w1] + self.vocab_size_unique)
        return laplace_dict

```

Література

- [1] Additive smoothing. https://en.wikipedia.org/wiki/Additive_smoothing.
- [2] Maximum likelihood estimation. https://en.wikipedia.org/wiki/Maximum_likelihood_estimation.
- [3] N-gram. <https://en.wikipedia.org/wiki/N-gram>.