3. **Языковая модель** N-**грамм** (n-gram language model)

Языковая модель

Статистическая языковая модель (language model, LM) - это модель, которая присваивает вероятности словам или последовательностям слов в целом.
N-грамма (n-gram) - это последовательность звуков, слогов, букв или слов из n элементов.
Применительно к ОЕЯ в роли элементов чаще всего выступают слова.
Зачем нам предсказывать последующие слова?
На самом деле, большие языковые модели, которые произвели революцию в современной ОЕЯ, обучаются просто путем предсказания слов!

Униграмма (unigram)

N-грамма, состоящая из одного элемента, называется **униграмма**.
На примере предложения "I am going fishing".

I	am	going	fishing
I	am	going	fishing
I	am	going	fishing
I	am	going	fishing

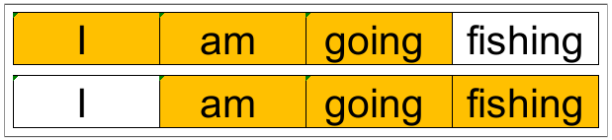
Биграмма (bigram)

N-грамма, состоящая из 2-х элементов, называется **биграмма**.
На примере предложения "I am going fishing".

I	am	going	fishing
I	am	going	fishing
I	am	going	fishing

Триграмма N-**грамм** (trigram)

N-грамма, состоящая из 3-х элементов, называется **триграмма**.
На примере предложения "I am going fishing".



Дополнение

Для того, чтобы мы могли получить истинное распределение вероятности необходимо добавить признаки начала и конца предложения.
На примере предложения "I am going fishing".

<s>	I	am	going	fishing	</s>
<s>	I	am	going	fishing	</s>
<s>	I	am	going	fishing	</s>
<s>	I	am	going	fishing	</s>
<s>	I	am	going	fishing	</s>

Алгоритм генерации N-грамм

In [1]:

```
n = 2

small_text = "I am going fishing"
tokens = small_text.split()

ngrams = [tuple(tokens[i:i + n]) for i in range(len(tokens) - n + 1)]

print(ngrams)
```

```
[('I', 'am'), ('am', 'going'), ('going', 'fishing')]
```

Генерация N-грамм с помощью NLTK

In [2]:

```
import nltk
from nltk.tokenize import word_tokenize
from nltk.util import ngrams
nltk.download('punkt', quiet=True)

small_text = "I am going fishing"
tokens = word_tokenize(small_text)

ngrams_list = ngrams(tokens, 2)

for ngram in ngrams_list:
    print(ngram)
```

```
('I', 'am')
('am', 'going')
('going', 'fishing')
```

Генерация N-грамм с помощью NLTK

Построим триграммы с признаками начала и конца предложения

In [3]:

```
import nltk
from nltk.tokenize import word_tokenize
from nltk.util import ngrams
nltk.download('punkt', quiet=True)

small_text = "I am going fishing"
tokens = word_tokenize(small_text)

ngrams_list = ngrams(tokens, 3, pad_left=True, pad_right=True,
                     left_pad_symbol='<s>', right_pad_symbol='</s>')

for ngram in ngrams_list:
    print(ngram)
```

```
('<s>', '<s>', 'I')
('<s>', 'I', 'am')
('I', 'am', 'going')
('am', 'going', 'fishing')
('going', 'fishing', '</s>')
('fishing', '</s>', '</s>')
```

Языковая модель N-**грамм** (n-gram language model)

Поговорим о вероятностях и начнем с задачи вычисления $P(w|h)$, т.е. вероятности слова w с учетом некоторой истории h. Пусть h имеет следующий вид: "он шёл домой, держа в руке", и мы хотим узнать вероятность того, что следующим словом будет "кофе".

$$P(\text{кофе}|\text{он шёл домой, держа в руке})$$

Как оценить такую вероятность?

Языковая модель N-**грамм** (n-gram language model)

Один из способов - это подсчет относительной частоты.

$$P(\text{кофе}|\text{он шёл домой, держа в руке}) = \frac{C(\text{он шёл домой, держа в руке кофе})}{C(\text{он шёл домой, держа в руке})},$$

где $C(\beta)$ - возвращает частоту β.

Языковая модель N-**грамм** (n-gram language model)

Сразу хочется проиндексировать всю сеть Интернет и вычислить относительные частоты. В целом это работает нормально, но как на счет такого?

$$P(\text{серрадуру}|\text{он шёл домой, держа в руке})$$

Ну или учитывая нестатичность ЕЯ:

$$P(\text{фейковый}|\text{он шёл домой, держа в руке})$$

Языковая модель N-**грамм** (n-gram language model)

Как вычислить вероятность $P(w_1, w_2, \ldots, w_n)$ последовательности слов $w_1, w_2, \ldots, w_n$ (для сокращения будем записывать $w_1, w_2, \ldots, w_n$ как $w_{1:n}$)?

Используя теорему умножения вероятностей получим следующее:

$$P(w_{1:n}) = P(w_1)P(w_2|w_1)P(w_3|w_{1:2}) \ldots P(w_n|w_{1:n-1}) = \\ = \prod_{k=1}^{n} P(w_k|w_{1:k-1})$$

Но вычислить $P(w_n|w_{1:n-1})$ мы не можем учитывая вышеозвученные ограничения.

Языковая модель N-**грамм** (n-gram language model)

Суть модели N-грамм заключается в том, что вместо вычисления вероятности слова, учитывая всю его историю, мы можем аппроксимировать историю только по n последним словам.
Предположение о том, что вероятность слова зависит только от предыдущего слова позволяют рассматривать языковую модель N-грамм как Марковский процесс.

Биграммы апроксимируют вероятность следующим образом:

$$P(w_n|w_{1:n-1}) \approx P(w_n|w_{n-1})$$

Для триграмм выражение будет имет вид:

$$P(w_n|w_{1:n-1}) \approx P(w_n|w_{n-2}, w_{n-1})$$

В общем виде для N-грамм:

$$P(w_n|w_{1:n-1}) \approx P(w_n|w_{n-N+1:n-1})$$

Языковая модель N-**грамм** (n-gram language model)

Используя биграмы мы можем вычислить вероятность для последовательности из n слов следующим образом:

$$P(w_{1:n}) \approx \prod_{k=1}^{n} P(w_k|w_{k-1})$$

Как оценить эти вероятности?

Языковая модель N-**грамм** (n-gram language model)

Метод максимального правдоподобия (maximum likelihood estimation, MLE)
Интуитивный способ оценки вероятностей называется методом максимального правдоподобия.
Мы применяем метод максимального правдоподобия для параметров N-граммной модели, получая значения из корпуса и нормализуя их на интервал $[0, 1]$.

Для вычисления вероятности биграммы $w_{n-1}w_n$, необходимо вычислить количество биграмм $C(w_{n-1}w_n)$ и нормализовать по сумме всех биграмм, начинающихся со слова w_{n-1}:

$$P(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)}{\sum_w C(w_{n-1}w)} = \frac{C(w_{n-1}w_n)}{C(w_{n-1})}$$

Языковая модель N-**грамм** (n-gram language model)

Метод максимального правдоподобия (maximum likelihood estimation, MLE)

Давайте рассмотрим пример:

<s> я преподаватель </s>

<s> кот это не я </s>

<s> я не люблю петь </s>

$P(\text{я}\|\text{<s>}) = \frac{2}{3}$	$P(\text{</s>}\|\text{преподаватель}) = 1$	$P(\text{я}\|\text{не}) = \frac{1}{2}$
$P(\text{кот}\|\text{<s>}) = \frac{1}{3}$	$P(\text{</s>}\|\text{я}) = \frac{1}{3}$	$P(\text{люблю}\|\text{не}) = \frac{1}{2}$

Языковая модель N-**грамм** (n-gram language model)

Метод максимального правдоподобия (maximum likelihood estimation, MLE)

Если рассматривать общий случай для N-грамм, то получим:

$$P(w_n|w_{n-N+1:n-1}) = \frac{C(w_{n-N+1:n-1}w_n)}{C(w_{n-N+1:n-1})}$$

Языковая модель N-**грамм** (n-gram language model)

Метод максимального правдоподобия (maximum likelihood estimation, MLE)

In [4]:

```
import nltk
nltk.download('punkt', quiet=True)
?
text = ''
with open(r'data/tolstoy.txt') as f:
    text = f.read().lower()

tokens = nltk.word_tokenize(text, language='russian')
vocabulary = list(set(tokens))
words = ['князь', 'андрей', 'как', 'будто', 'хотел', 'сказать']

fdist = nltk.FreqDist(tokens)
fdist.tabulate(samples=words)

cfd = nltk.ConditionalFreqDist(nltk.bigrams(tokens))
cfd.tabulate(conditions=words, samples=words)
```

```
  князь  андрей     как   будто   хотел сказать
   1328     793    4003     479     243     251
         князь  андрей     как   будто   хотел сказать
князь        0     777       1       0       0       0
андрей       0       0       0       0       4       0
   как      24       0       0     430       1       2
будто        1       0       0       0       1       0
```

хотел	0	0	0	0	0	23
сказать	0	0	0	0	0	0

Языковая модель N-**грамм** (n-gram language model)

Метод максимального правдоподобия (maximum likelihood estimation, MLE)

In [2]:

```
import nltk
from nltk.lm.preprocessing import padded_everygram_pipeline
from nltk.lm import MLE
nltk.download('punkt', quiet=True)

text = ''
with open(r'data/tolstoy.txt') as f:
    text = f.read().lower()

tokens = [nltk.word_tokenize(sentence, language='russian')
          for sentence in nltk.sent_tokenize(text)]
train, vocab = padded_everygram_pipeline(2, tokens)

model = MLE(order=2)
model.fit(train, vocab)

words = ['князь', 'андрей', 'как', 'будто', 'хотел', 'сказать']
for w1 in words:
    for w2 in words:
        print(model.counts[[w1]][w2], end='\t')
    print()
```

```
0       777     1       0       0       0
0       0       0       0       4       0
24      0       0       430     1       2
1       0       0       0       1       0
```

0	0	0	0	0	23
0	0	0	0	0	0

Относительные частоты

Подсчитаем частоты униграмм:

князь	андрей	как	будто	хотел	сказать
1328	793	4003	479	243	251

Подсчитаем частоты биграмм:

	князь	андрей	как	будто	хотел	сказать
князь	0	777	1	0	0	0
андрей	0	0	0	0	4	0
как	24	0	0	430	1	2
будто	1	0	0	0	1	0
хотел	0	0	0	0	0	23
сказать	0	0	0	0	0	0

Языковая модель N-**грамм** (n-gram language model)

Метод максимального правдоподобия (maximum likelihood estimation, MLE)

In [9]:

```
words = ['князь', 'андрей', 'как', 'будто', 'хотел', 'сказать']
for w1 in words:
    print(w1, end='\t')
    for w2 in words:
        print(round(model.score(w2, context=[w1]), 5), end='\t')
    print()
```

```
князь   0.0     0.58509 0.00075 0.0     0.0     0.0
андрей  0.0     0.0     0.0     0.0     0.00504 0.0
как     0.006   0.0     0.0     0.10742 0.00025 0.0005
будто   0.00209 0.0     0.0     0.0     0.00209 0.0
хотел   0.0     0.0     0.0     0.0     0.0     0.09465
сказать 0.0     0.0     0.0     0.0     0.0     0.0
```

Относительные частоты

Подсчитаем относительные частоты:

	князь	андрей	как	будто	хотел	сказать
князь	0	0.58509	0.000753	0	0	0
андрей	0	0	0	0	0.005044	0
как	0.005996	0	0	0.107419	0.00025	0.0005
будто	0.002088	0	0	0	0.002088	0
хотел	0	0	0	0	0	0.09465
сказать	0	0	0	0	0	0

Относительные частоты

$$P(\text{я хотел тебе сказать}) = P(\text{хотел}|\text{я})P(\text{тебе}|\text{хотел})P(\text{сказать}|\text{тебе}) \approx 0.0035 \cdot 0.0041 \cdot 0.03 = 0.000000431$$

- Что будет происходить, если предложение окажется хоть чуточку длиннее?
- А если вероятность какой-либо биграммы окажется нулевой?

Логарифмирование

$$\prod_{i=1}^{N} p_i = \exp\left(\sum_{i=1}^{N} \ln p_i\right)$$

Оценка языковых моделей

- **Внешняя оценка** - оценивание модели путём решения с её помощью поставленно задачи. Это лучший подход к оцениванию моделей, но такой подход медленный и может потребовать больших вычислительных ресурсов.
- **Внутренняя оценка** - оценивание модели путём использования некоторой метрики, без учёта конкретных задач, для решения которых модель планируется использовать. Такая оценка хуже внешней, но это быстрый и полезный инстркмент оценки и сравнения моделей.

Оценка языковых моделей

Наборы данных:

- **Обучающий** - это данные, которые мы используем для обучения параметров нашей модели. Для простых моделей языка типа N-грамм это корпус, из которого мы получаем вероятности.
- **Тестовый** - это ограниченный набор данных, не пересекающийся с обучающим набором, который мы используем для оценки модели.

Все наборы должны быть репрезентативными по отношению к решаемой задаче.

Оценка языковых моделей: **перплексия** (perplexity, PP, PPL)

Это одна из наиболее важных метрик используемых в ОЕЯ.

$$\text{perplexity}(W) = P(w_1 w_2 \dots w_N)^{-\frac{1}{N}} = \sqrt[N]{\frac{1}{P(w_1 w_2 \dots w_N)}}$$

Используя теорему умножения вероятностей получим следующее:

$$\text{perplexity}(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i | w_1 w_2 \dots w_{i-1})}}$$

Следует обратить внимание, что перплексия обратная вероятностям величина.

Оценка языковых моделей: **перплексия** (perplexity, PP, PPL)

Для униграмм перплексия вычисляется следующим образом:

$$\text{perplexity}(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i)}}$$

Для биграмм:

$$\text{perplexity}(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i|w_{i-1})}}$$

Оценка языковых моделей: **перплексия** (perplexity, PP, PPL)

Перплексию можно рассматривать как **коэффициент ветвления языка**. Коэффициент ветвления языка - это количество возможных последующих слов, после любого слова языка. Для примера возьмем автоматный язык целых чисел, в котором после каждой цифры может следовать последующая цифра. Считаем, что вероятности встретить любую из цифр равны. Если имеем число, состоящее из N цифр, то получим:

$$\text{perplexity}(W) = P(w_1 w_2 \dots w_N)^{-\frac{1}{N}} = \left(\frac{1}{10}^{N}\right)^{-\frac{1}{N}} = 10$$

Генерация предложений

Этот метод впервые был предложен Шенноном (1948), Миллером и Селфриджем (1950).

1. Сначала создаем случайную биграмму, которая начинается с <s> (в соответствии с вероятностью ее биграммы).
2. Выбираем второе слово полученной биграммы - w.
3. Затем мы выбираем случайную биграмму, начинающуюся с w (в соответствии с вероятностью ее биграммы).
4. Повторяем шаг 2.

Обобщение

- Модель N-грамм (как и многие статистические модели) зависит от обучающего корпуса.
- Модель N-грамм лучше моделирует обучающий корпус по мере того, как мы увеличиваем значение N.

Обобщение

- **Униграмма** "наивным и наташа алмазным себя сказал он ? летать голоса в чувствовал — , страшный".
- **Биграмма** "потом искусные маневры , укоризненное лицо козловского , обошел кабинет , — я богата ?".
- **Триграмма** "граф илья андреич беспрестанно ездил по полю , как есть офицер , — была смоленск".
- **4-грамма** "<s> <s> предчувствие анны павловны оправдалось , и в лице его отдавалась честь боевому ,".

Неизвестные слова (**проблема** out of vocabulary, OOV).

Что делать со словами, которые мы никогда раньше не встречали?
Использовать открытый словарь. Можно завести специальный тег для неизвестного слова <UNK>, возвращая задачу к задаче с закрытым словарем, заранее выбирая фиксированный словарный запас:

1. Выбераем фиксированный словарь.
2. Если встречаем слово, которого не было в обучающем наборе, то преобразуем его в токен неизвестного слова <UNK> на этапе нормализации текста.
3. Оцениваем вероятности для <UNK>, как и для любого другого обычного слова из обучающего набора.

Нули

Что делать, если:

$$P(w_i|w_{i-1}) = 0$$

Такого не может происходить при работе с обучающим набором данных, но в тестовом наборе нули представляют проблему:

1. Их присутствие означает, что мы недооцениваем вероятность появления всех видов слов.
2. Если вероятность любого слова из тестовых данных равна 0, вся вероятность набора слов из тестовых данных равна 0.

Сглаживание (smoothing)

Что с этим всем делать?

	князь	андрей	как	будто	хотел	сказать
князь	0	0.58509	0.000753	0	0	0
андрей	0	0	0	0	0.005044	0
как	0.005996	0	0	0.107419	0.00025	0.0005
будто	0.002088	0	0	0	0.002088	0
хотел	0	0	0	0	0	0.09465
сказать	0	0	0	0	0	0

Сглаживание Лапласа (Laplace smoothing)

Очень простое решение - добавить к частоте всех N-граммам $+1$. Работает такой подход плохо, но позволяет познакомится с общей концепцией.
Для униграмм слова выглядит как:

$$P(w_i) = \frac{c_i}{|D|},$$

где c_i - счетчик слова, отнормированный на общее количество слов $|D|$.
Тогда сглаживание Лапласа для униграмм (additive smoothing) будет выглядить как:

$$P_L(w_i) = \frac{c_i + 1}{|D| + |\mathcal{V}|},$$

Сглаживание Лапласа

Чтобы не пересчитывать каждый раз числитель и знаменатель, для удобства определим:

$$c_i^* = (c_i + 1)\frac{|D|}{|D| + |\mathcal{V}|}$$

Теперь мы можем рассчитать P_i^*:

$$P_i^* = \frac{c_i^*}{|D|}$$

И ввести понятие относительного дисконта:

$$d = \frac{c^*}{c}$$

Сглаживание Лапласа

Теперь можно сгладить биграммы в соответствии с:

$$P_L(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n) + 1}{\sum_w (C(w_{n-1}w) + 1)} = \frac{C(w_{n-1}w_n) + 1}{C(w_{n-1}) + |\mathcal{V}|}$$

Сглаживание Лапласа

In [10]:

```
import nltk
from nltk.lm.preprocessing import padded_everygram_pipeline
from nltk.lm import Laplace
nltk.download('punkt', quiet=True)

text = ''
with open(r'data/tolstoy.txt') as f:
    text = f.read().lower()

tokens = [nltk.word_tokenize(sentence, language='russian') for sentence in nltk.sent_tokenize(text)]
train, vocab = padded_everygram_pipeline(2, tokens)

model = Laplace(order=2)
model.fit(train, vocab)
```

In [11]:

```
words = ['князь', 'андрей', 'как', 'будто', 'хотел', 'сказать']
for w1 in words:
    print(w1, end=' ')
    for w2 in words:
        print(model.score(w2, context=[w1]), end='\t')
    print()
```

```
князь 1.845188670541563e-05     0.01435556785681336     3.690377341083126e-05   1.845188670541
563e-05 1.845188670541563e-05   1.845188670541563e-05
андрей 1.8635855385762205e-05   1.8635855385762205e-05  1.8635855385762205e-05  1.863585538576
2205e-05        9.317927692881103e-05   1.8635855385762205e-05
как 0.0004395990856339018       1.7583963425356075e-05  1.7583963425356075e-05  0.007578688236
328468  3.516792685071215e-05   5.275189027606823e-05
```

будто 3.7491095864732125e-05 1.8745547932366062e-05 1.8745547932366062e-05 1.874554793236
6062e-05 3.7491095864732125e-05 1.8745547932366062e-05
хотел 1.8828845791752966e-05 1.8828845791752966e-05 1.8828845791752966e-05 1.882884579175
2966e-05 1.8828845791752966e-05 0.00045189229900207117
сказать 1.8826010015437328e-05 1.8826010015437328e-05 1.8826010015437328e-05 1.882601001543
7328e-05 1.8826010015437328e-05 1.8826010015437328e-05

Сглаживание Лапласа

	князь	андрей	как	будто	хотел	сказать
князь	1	778	2	1	1	1
андрей	1	1	1	1	5	1
как	25	1	1	431	2	3
будто	2	1	1	1	2	1
хотел	1	1	1	1	1	24
сказать	1	1	1	1	1	1

Сглаживание Лапласа

Вероятности сильно изменились.

	князь	андрей	как	будто	хотел	сказать
князь	0	0.58509	0.000753	0	0	0
андрей	0	0	0	0	0.005044	0
как	0.005996	0	0	0.107419	0.00025	0.0005
будто	0.002088	0	0	0	0.002088	0
хотел	0	0	0	0	0	0.09465
сказать	0	0	0	0	0	0

	князь	андрей	как	будто	хотел	сказать
князь	0.000018	0.014356	0.000037	0.000018	0.000018	0.000018
андрей	0.000019	0.000019	0.000019	0.000019	0.000093	0.000019
как	0.000440	0.000018	0.000018	0.007579	0.000035	0.000053
будто	0.000037	0.000019	0.000019	0.000019	0.000037	0.000019
хотел	0.000019	0.000019	0.000019	0.000019	0.000019	0.000452
сказать	0.000019	0.000019	0.000019	0.000019	0.000019	0.000019

Сглаживание Лапласа

Попробуйте пересчитать самостоятельно относительный дисконт.

Сглаживание Лапласа

Можно восстановить частоты:

$$\mathrm{c}^*(w_{n-1}w_n) = \frac{(C(w_{n-1}w_n)+1)\cdot C(w_n)}{C(w_{n-1})+|\mathcal{V}|}$$

Add-k **сглаживание** (lidstone smoothing)

Одна из альтернатив, это добавление не +1, а некоторого, небольшого дробного числа k (0.5, 0.05, 0.01).

$$P_{\text{Add-k}}(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n) + k}{C(w_{n-1}) + k|\mathcal{V}|}$$

Для выбора k, требуется дополнительный набор данных. Метод add-k сглаживания, как и сглаживание Лапласа, не очень хорош.

Add-k **сглаживание**

In [9]:

```
import nltk
from nltk.lm.preprocessing import padded_everygram_pipeline
from nltk.lm import Lidstone
nltk.download('punkt', quiet=True)

text = ''
with open(r'data/tolstoy.txt') as f:
    text = f.read().lower()

tokens = [nltk.word_tokenize(sentence, language='russian') for sentence in nltk.sent_tokenize(text)]
train, vocab = padded_everygram_pipeline(2, tokens)

model = Lidstone(order=2)
model.fit(train, vocab)
```

Откат и интерполяция

- Откат (backoff) - переход к N-граммам с более низким n ($3 \to 2 \to 1$).
- Интерполяция (interpolation) - смешивание оценок N-грамм с разными n (триграммы, биграммы, униграммы).

Откат

- Пытаемся вычислить $P(w_n|w_{n-2}w_{n-1})$, но примеров $w_{n-2}w_{n-1}w_n$ не было, тогда оцениваем вероятность $P(w_n|w_{n-1})$.
- Аналогично, если у нас нет примеров для вычисления $P(w_n|w_{n-1})$, оцениваем вероятность $P(w_n)$.

Линейная интерполяция

Суммирование оценок триграмм, биграмм и униграмм с соответствующими коэффициентами λ_i.

$$\hat{P}(w_n|w_{n-2}w_{n-1}) = \lambda_1 P(w_n) + \lambda_2 P(w_n|w_{n-1}) + \lambda_3 P(w_n|w_{n-2}w_{n-1})$$

При этом $\sum_i \lambda_i = 1$.

Условная интерполяция

$$\hat{P}(w_n|w_{n-2}w_{n-1}) = \lambda_1(w_{n-2}:w_{n-1})P(w_n) + \lambda_2(w_{n-2}:w_{n-1})P(w_n|w_{n-1}) + \lambda_3(w_{n-2}:w_{n-1})P(w_n|w_{n-2}w_{n-1})$$

Можно подбирать λ_i на отложеных данных, оптимизируя по перплексии.