Търсене и извличане на информация. Приложение на дълбоко машинно обучение

Зимен семестър 2021/2022

Домашно задание №1

20.11.2021

Общ преглед

В това задание ще имплементираме вероятностен коректор на правописа, за да коригираме автоматично евентуалните правописни грешки в заявките. Разглеждаме процес при който потребителя желае да изпише изказ q, но вместо това изписва изказ r, който евентуално съдържа грешки. С $\mathrm{T(q, r)}$ ще бележим множеството от всички вериги от елементарни операции $op \in \mathrm{Op}$, които съответстват на изписан изказ r при желан изказ q. Формално дефинираме множеството от елементарни операции Op над азбука от символи Σ като:

$$\mathrm{Op} = \mathrm{Id} \cup \mathrm{Ins} \cup \mathrm{Del} \cup \mathrm{Sub} \cup \mathrm{Trans},$$

където

$$\begin{aligned} \mathrm{Id} &= \{(\sigma, \sigma) | \sigma \in \Sigma\} \\ \mathrm{Ins} &= \{(\varepsilon, \sigma) | \sigma \in \Sigma\} \\ \mathrm{Del} &= \{(\sigma, \varepsilon) | \sigma \in \Sigma\} \\ \mathrm{Sub} &= \{(\sigma, \tau) | \sigma, \tau \in \Sigma, \sigma \neq \tau\} \\ \mathrm{Trans} &= \{(\sigma\tau, \tau\sigma) | \sigma, \tau \in \Sigma, \sigma \neq \tau\}. \end{aligned}$$

Ако $op = (x, y)$ то $op_1 = x, op_2 = y$. Ще оценим вероятността на дадена верига t от елементарни операции с произведението от вероятностите за всяка операция от нея. Тоест ще предположим, че процеса е

породен от монограмен марковски модел на ниво елементарни операции. Нека $t = (op^1, op^2, \ldots, op^k)$, тогава $\Pr[t] = \prod_{i=1}^{k} \Pr[op^i]$. За краткост $t_1 = op_1^1 op_1^2 \ldots op_1^k$ и $t_2 = op_2^1 op_2^2 \ldots op_2^k$, тогава от $t \in \mathrm{T(q,r)}$ следва, че $t_1 = q$ и $t_2 = r$. Крайната цел на нашето задание е да намерим най-вероятния изказ q^r разполагайки само със сгрешения изказ r. Избираме q^r за най-добър кандидат за желаната заявката, където $q^r = t_1^r$ и:

$$t^r = \arg\max_{t \in T(\cdot,r)} \Pr[t]$$

$$\mathrm{T}(\cdot, \mathrm{r}) = \bigcup_{\mathrm{q} \in \Sigma^*} \mathrm{T}(\mathrm{q}, \mathrm{r})$$

Забележка: На практика няма да извършим обединението $\bigcup \mathrm{T(q,r)}$ по всички думи $q \in \Sigma^*$, а само по $q \in \Sigma^* \cap D^*$, където D е предварително избрано ограничаващо множество. В нашия случай рестрикцията ще бъде върху думите от подбран речник съдържащ само правилни думи.

За приближение на вероятноста $\Pr[op]$ ще използваме емпиричната вероятност $\hat{\Pr}[op]$. Ще извлечем емпиричната вероятност от подравняване на сгрешен корпус с правилен корпус, като приемем, че правилният корпус е изказа, който потребителя е желал да изпише, а сгрешения корпус е действително изписаният. Така имайки подравняването извличаме монограмния марковски модел. Имайки модела ще можем да коригираме и произволни сгрешени корпуси.

Предвид горната формулировка, ще изградим вероятностен коректор на правописа, на няколко етапа:

1. Подравнител. По желан изказ q и сгрешен изказ r избира подходящ елемент на $\mathrm{T(q,r)}$.

2. Оценител. Ще оценим вероятността на всяка една от елементарните правописни грешки, които могат да възникнат при изписването на заявката. По-конкретно, символите в дадена заявката да бъдат изтрити по погрешка, вмъкнати, заместени, транспонирани или запазени непроменени.

3. Генератор на кандидати. По оригиналната заявка r, зададена от потребителя, се генерира множество от кандидати за желаната заявка q.

4. Коригиращ модел. Комбинирайки 1, 2 и 3, ще намерим най-добрия кандидат за желаната заявка.

1 Разстояние на Левенщайн - Дамерау

Първата задача е да се имплементира функция, изчисляваща разстоянието на Левенщайн - Дамерау между две думи. На първата лекция разгледахме дефиницията на Левенщайн разстояние и псевдокод на алгоритъм, който го изчислява. Разстоянието на Левенщайн - Дамерау е подобно на Левенщайн разстоянието, но като елементарни операции се допускат освен изтриване, вмъкване и субституции на символи, също и транспозиция на два съседни символа. По-формално:

$$
\begin{aligned}
d_L(\varepsilon, \varepsilon) &= 0 \\
d_L(\varepsilon, Wa) &= d_L(\varepsilon, W) + 1 \\
d_L(Pa, \varepsilon) &= d_L(P, \varepsilon) + 1 \\
d_L(Pa, Wb) &= \begin{cases}
\min(& d_L(P, W) + \delta_{a \neq b}, \\
 & d_L(Pa, W) + 1, \\
 & d_L(P, Wb) + 1, \\
 & d_L(P', W') + 1) & \text{if } P = P'b \ \&\ W = W'a \\
\min(& d_L(P, W) + \delta_{a \neq b}, \\
 & d_L(Pa, W) + 1, \\
 & d_L(P, Wb) + 1) & \text{otherwise}
\end{cases}
\end{aligned}
,
$$

където $\delta_{a \neq b} = 1$, ако $a \neq b$, и $\delta_{a \neq b} = 0$ в противен случай.

Например: Заявката „Иван обича Мария много“ е на разстояние 7 от „Иванн убичъ аМрия мн“. Седем на брой са грешките - нетривиалните операции. Всяка от думите се получава посредством следните операции:

$$
\begin{aligned}
\text{Ops\_}[\text{иван}, \text{иванн}] &= \{(\text{и}, \text{и}), (\text{в}, \text{в}), (\text{а}, \text{а}), (\text{н}, \text{н}), (\varepsilon, \text{н})\} \\
\text{Ops\_}[\text{обича}, \text{убичъ}] &= \{(\text{о}, \text{у}), (\text{б}, \text{б}), (\text{и}, \text{и}), (\text{ч}, \text{ч}), (\text{а}, \text{ъ})\} \\
\text{Ops\_}[\text{мария}, \text{амрия}] &= \{(\text{ма}, \text{ам}), (\text{р}, \text{р}), (\text{и}, \text{и}), (\text{я}, \text{я})\} \\
\text{Ops\_}[\text{много}, \text{мн}] &= \{(\text{м}, \text{м}), (\text{н}, \text{н}), (\text{о}, \varepsilon), (\text{г}, \varepsilon), (\text{о}, \varepsilon)\}
\end{aligned}
$$

Където ε съответства на празен низ.

За да имплементирате ефективен алгоритъм, намиращ разстоянието на Левенщайн-Дамерау съгласно горепосочената формула е препоръчително да използвате метод на динамичното програмиране. За подравняване на два низа ще използваме пътя, по който е получено разстоянието на Левенщайн-Дамерау. Вместо сумата от дължините (0 за $op \in \mathrm{Id}$ и 1 иначе) на съответните (минимално на брой) операции, ще вземем конкатенацията на самите операции. Можете да имплементирате това ефективно, като във всяка клетка на динамичната таблица си запазите и съответната операция и предходната клетка, за да можете да възстановите целия път, траверсирайки таблицата от края към началото.

Задачата е да попълните тялото на функциите `editDistance` и `editOperations` в кода на програмата (2.5 т.)

2 Оценител – тегло на редакция

Втората задача е свързана с оценяването на вероятността на всяка елементарна операция и теглото на редакцията за получаване на една дума от друга. Както отбелязахме ще извлечем емпиричните вероятности от подравняването на два корпуса - правилен C_0 и сгрешен C_1. Подравняването на несгрешените думи е очевидно - композиция от идентитети за всеки символ. За подравняване на сгрешените думи ще използваме подравнителя от точка 1. Корпусите са така подготвени, че да няма грешки при подравняването на ниво думи. Думата с пореден индекс i от C_0 съответства на i-тата дума от C_1, с точност до реалистично подравняване. Тоест подравняването ни е осигурено на ниво думи и ние трябва да се погрижим само за подравняването на ниво символи в рамките на всяка дума. След това, имайки двата корпуса C_0 и C_1 представени като верига $c \in T(C_0, C_1)$, можем да извлечем монограмен марковски модел и да изчислим емпиричните вероятности за всяка отделна елементарна операция. А именно:

$$\hat{Pr}[op] = \frac{\alpha + \#(op)}{\alpha * |\mathrm{Op}| + \sum\limits_{op \in \mathrm{Op}} \#(op)}$$

Където α е коефициент на изглаждане, а $\#(\cdot)$ е брой срещания в корпуса.

Ще предполагаме, че ни е дадена функция $\omega : \mathrm{Op} \to \mathbb{R}^+$, която на всяка елементарна операция ни съпоставя тегло. Ще предполагаме, че $\omega(op) = -\log \hat{\Pr}[op]$. Тогава теглото на по-сложна редакция - верига от

елементарни операции - съответства на минус логаритъм от вероятността т.е. $\omega(t) = -\log \hat{\Pr}[t]$, за $t \in T(q,r)$. Ако $t = (op^1, op^2, \ldots, op^k)$, то $\omega(t) = -\log \prod_{i=1}^{k} \hat{\Pr}[op^i] = \sum_{i=1}^{k} -\log \hat{\Pr}[op^i] = \sum_{i=1}^{k} \omega(op^i)$

Така от емпиричните вероятности получаваме и теглата на елементарните операции. Например $\omega(op)$ се очаква да бъде по-близко до 0 за $op \in \mathrm{Id}$, тъй като ще имаме значително повече тривиални редакции - т.е. идентитет - отколкото същински грешки.

Дефинираме функцията Γ, която следва да връща минималното тегло на подравняване на думата r (съответстваща на изписаната дума) с думата q (съответстваща на желаната дума) индуктивно:

$$
\begin{aligned}
\Gamma(\varepsilon,\varepsilon) &= 0 \\
\Gamma(\varepsilon,Wa) &= \Gamma(\varepsilon,W) + \omega(\varepsilon,a) \\
\Gamma(Pa,\varepsilon) &= \Gamma(P,\varepsilon) + \omega(a,\varepsilon) \\
\Gamma(Pa,Wb) &= \begin{cases}
\min(& \Gamma(P,W) + \omega(a,b), & \\
 & \Gamma(Pa,W) + \omega(\varepsilon,b), & \\
 & \Gamma(P,Wb) + \omega(a,\varepsilon), & \\
 & \Gamma(P',W') + \omega(ba,ab)) & \text{if } P = P'b \ \&\ W = W'a \\
\min(& \Gamma(P,W) + \omega(a,b), & \\
 & \Gamma(Pa,W) + \omega(\varepsilon,b), & \\
 & \Gamma(P,Wb) + \omega(a,\varepsilon)) & \text{otherwise}
\end{cases}
\end{aligned}
$$

Забележка: Функцията Γ не е симетрична спрямо аргументите си.

Докажете, че:

$$e^{\Gamma(q,r)} = \max_{t \in T(q,r)} \hat{\Pr}[t]$$

Помощно: докажете, че $\Gamma(q,r)$ връща най-малкото тегло на подравняване на думата r с думата q. (3 т.)

Попълнете тялото на функцията `computeOperationWeights` в кода на програмата. (1 т.)

Попълнете тялото на функцията `editWeight` в кода на програмата, така че да имплементира функцията Γ (1 т.)

Забележка: За осигуряване на по-добра числова стабилност е желателно навсякъде вместо стойностите на вероятностите да се използват логаритъм от съответните вероятности. Тъй като логаритъмът е монотонно разстяща функция имаме, че $\arg\max p = \arg\max \log p$. Освен това $\log \prod_i p_i = \sum_i \log p_i$.

3 Генератор на кандидати

Тъй като почти всички правописни грешки се намират в рамките на разстояние до 2 от желаната от потребителя заявка, ще търсим кандидатите за корекции на заявки на разстояние до 2 от r. За заявки r с нетривиална дължина, обаче, броят на кандидатите става огромен. Има различни подходи за ефективно генериране на кандидати, но предлагаме да се приложи следната идея: Започнете, като генерирате всички възможни редакции, които са на разстояние 1 от оригиналната заявка. За всяка редакция на разстояние 1 от оригиналната отново да извикаме функцията, за да генерираме кандидатите на разстояние две. Накрая ще изискваме всички думи измежду кандидатите да бъдат от речника. Корпуса е подбран така, че е достатъчно нашата азбука се състои само от буквите от българската азбука. Интервал, тире, както и други пунктуационни символи могат да не участват в азбуката, тъй като в корпусите не са допуснати грешки с тях.

Има, разбира се, други, много по-ефективни стратегии и много възможни разширения и вариации на стратегията, спомената тук.

Попълнете тялото на функциите `generateEdits, generateCandidates` в кода на програмата, така че по зададена оригинална заявка да се генерират всички заявки, които са на Левенщайн - Дамерау разстояние до 2 от оригиналната и се състоят единствено от думи от речника на корпуса (1.5 т.)

4 Коригиращ модел

Работата на коригиращия модел е да намери най-вероятната заявка q по дадено сгрешено изписване r. За получаването на кандидатите за q ще използваме генератора на кандидати. Формално, при дадена оригинална заявка r търсим q:

$$q = \arg\min_{q \in C_r} \omega(q, r)$$

Където C_r множество от възможни заявки. Моделирайки вероятностите за операциите с монограмен модел, ние приемаме, че отделните операции са независими. В частност последователните вериги са независими една от друга. Това ни позволява при многословна заявка r, да разгледаме думите на r една по една и да намерим наи-вероятна верига за всяка една от тях, като в последствие комбинираме резултата. Ко-

гато заявката r се състои от една дума, ще използваме генератора на кандидати за да получим C_r.

Попълнете тялото на функцията `correctSpelling` в кода на програмата, така че да имплементира функция за оценяване на кандидатите (1 т.)

Инструкция за предаване на домашна работа

Изисква се в Moodle да бъде предаден архив FNXXX.zip (където XXX е вашият факултетен номер), който съдържа:

1. Файл a1.py, съдържащ нанесените от вас промени
2. Доказателство на твърдението от точка 2 в условието. Доказателството може да е под форма на сканирано/снимано доказателство на хартия или pdf.
3. Файловете `probabilities.pkl` и `corrected.txt` - създадени чрез изпълнението на `python3 test.py` .