EP1 - MODELOS GENERATIVOS DE LINGUAGEM

1. Motivação e objetivos

Neste EP, você desenvolverá um sistema de geração de textos em português baseado em modelos de linguagem markovianos. Pense nesse sistema como o tataravô do ChatGPT.

Um *modelo de linguagem* associa um valor de probabilidade $\Pr(w_1, \ldots, w_s)$ a cada sentença $w_1, \ldots, w_s$ de uma linguagem. Por exemplo, um modelo de linguagem em português pode associar uma probabilidade $\Pr(\text{hoje}, \text{está}, \text{quente})$ a sentença 'hoje está quente' mais alta que a probabilidade $\Pr(\text{hoje}, \text{está}, \text{insalubre})$. Tais probabilidades podem ser *aprendidas* ou estimadas a partir de um corpus[1] volumoso de sentenças.

Dado um modelo de linguagem, geramos um texto selecionando uma frase com alta probabilidade, possivelmente entre aquelas que complementam um dado texto inicial. Sua tarefa nesse EP será formular o problema de geração de texto como um problema de busca em espaço de estados, a ser resolvido por algoritmos genéricos de busca heurística e local.

2. Modelo markoviano de linguagem

De acordo com as regras da teoria de probabilidades, uma probabilidade *conjunta* pode ser escrita como o produto de probabilidades condicionais, que no caso do modelo linguagem nos permite especificar a probabilidade conjunta baseado apenas na probabilidade $\Pr(w_i|w_1, \ldots, w_{i-1})$ de predição da próxima palavra condicional nas palavras anteriores:

$$\Pr(w_1, \ldots, w_s) = \Pr(w_1) \times \Pr(w_2|w_1) \times \cdots \times \Pr(w_s|w_1, \ldots, w_{s-1}).$$

Um modelo markoviano de ordem N assume que cada evento w_i é condicionalmente probabilisticamente independente de seus eventos passados dados os N eventos anteriores. Em linguagem matemática, temos que:

$$\Pr(w_i|w_1, \ldots, w_{i-1}) = \Pr(w_i|w_{i-1}, \ldots, w_{i-N}).$$

Embora tal hipótese seja claramente falsa pra linguagens humanas, ela provê uma aproximação razoável para valores altos de N.

O modelo de linguagem que assume a hipótese markoviana de ordem N acima é chamado de modelo de n-gramas, com $n = N + 1$. Por exemplo, um modelo de 2-gramas (chamado de bigramas) usa um modelo markoviano de ordem 1: $\Pr(w_i|w_{i-1})$.

A primeira etapa é segmentar um corpus em sentenças. Cada sentença é então segmentada em um conjunto de elementos constituintes chamados de sintagmas (ou *tokens*, em inglês), o equivalente a palavras. Essas tarefas são menos triviais do que podem parecer à primeira vista, porém para esse exercício vamos adotar uma estratégia simples de uso de expressões regulares (os modelos mais avançados

Date: April 12, 2024.

[1]Um corpus é uma coleção de textos reunidos para fins de alguma análise linguística.

resolvem um problema de otimização para chegar ao melhor segmentador), que por exemplo identificam palavras como sequências de caracteres separados por espaços ou pontuação e sentenças como sequências terminadas por pontuação ou quebra de linha.

A segunda etapa é então estimar as probabilidades de predição da próxima palavra:

$$\Pr(w_i|w_{i-1},\ldots,w_{i-N}) = \frac{N(w_{i-N},\ldots,w_i)}{N(w_{i-N},\ldots,w_{i-1})},$$

onde $N(s)$ é a contagem do N-gramas s em algum corpus. Note que para $i < N$, a função acima é aplicada em M-gramas com $M < N$. Para evitar ter que lidar com funções distintas para $i < N$ e $i \geq N$, nós prefixamos cada sentença com $N-1$ indicadores de início `<s>`, de forma que a probabilidade das palavras de início da frase podem ser também estimadas como N-gramas. Por exemplo, para um modelo de ordem 1 (portanto $n = 2$), sufixamos cada sentença com um indicador `<s>`, de forma que $\Pr(w_1)$ é identificado como $\Pr(w_1| < \text{s} >)$. O término de cada sentença é representado pelo indicador `</s>`. Assim, um corpus com o seguinte conteúdo

> Esse é um texto de exemplo
> Ele é composto de duas sentenças

é segmentado para produzir os seguintes bigramas: (<s>, Esse), (Esse, é), (é, um), (um, texto), (texto, de), (de, exemplo), (exemplo, </s>), (<s>, Ele), (Ele, é), (é, composto), (composto, de), (de, duas), (duas, sentenças), (sentenças, </s>).

Para sua conveniência, é fornecida uma classe `LanguageModel` que realiza as etapas de segmentação de um corpus (no formato de um arquivo de texto) em N-gramas e devolve a contagem de cada N-grama.

3. Representação como problema de busca heurística

Sua primeira tarefa é formular o problema de geração de uma sentença como busca em espaço de estados. Você deve implementar a classe `CompleteSentence`, que especializa a interface `Problem` para representar o problema de busca. Em particular, você deve decidir o que é uma representação adequada para o estado e implementar as funções de estado inicial, transição, ações aplicáveis, custo e teste de meta. O estado inicial representa uma sentença parcial, a qual queremos gerar um texto que complete a frase dada (para gerar um texto qualquer, basta considerar um texto vazio como estado inicial). As ações aplicáveis devem devolver as palavras que possuem probabilidade positiva para o contexto de $N-1$ últimas palavras do texto parcialmente gerado. A função de transição acrescenta uma nova palavra (ação) à sentença parcialmente gerada. O teste de meta deve verificar se o identificador de término de frase `</s>` foi gerado. O custo de transição deve ser dado pelo negativo do logaritmo da probabilidade de acrescentar uma palavra w_i a uma sentença que termina com as palavras $w_{i-1},\ldots,w_{i-1}$, considerando um modelo de ordem N: $-\log\Pr(w_i|w_{i-1},\ldots,w_{i-N})$.

Sua implementação deve funcionar nos algoritmos de busca fornecidos (busca por custo uniforme, busca em profundidade limitada e busca em feixe), sem alterá-los. A busca em feixe é um tipo de busca local que mantém uma lista de K estados. A cada iteração, cada candidato dessa lista é expandido, gerando novos estados para cada estado da lista. Os K estados de menor custo são mantidos para a próxima iteração. Caso um estado meta esteja entre os K mantidos, ele é retirado da lista e adicionado a uma lista de soluções. O algoritmo só para quando a lista de soluções

possuir K elementos ou não houver mais estados na fronteira. Assim como a busca em profundidade, a busca em feixe não garante otimalidade. Porém, na prática observa-se que ela desempenha bem para valores razoáveis de K.

Execute alguns experimentos usando um modelo de trigramas ($N = 2$), variando o estado inicial e algoritmo de busca.

4. Probabilidade e Perplexidade

Na definição do problema apresentada até agora, cada palavra adicionada a uma sentença parcial s gera uma nova sentença s' com duas propriedades: s' é mais comprida que s e a probabilidade de s' é igual ou menor que a probabilidade de s. Em outras palavras, o custo de gerar um sentença aumenta com o tamanho da sentença. Isso faz com que as buscas encontrem sentenças curtas e não muito interessantes como solução, e que busca subótimas tendam a gerar sentenças mais interessantes que buscas ótimas.

Uma maneira de aliviar o papel do comprimento da frase em sua avaliação é substituir o negativo da log-probabilidade como função de custo pela **perplexidade** (PPL), dada por:

$$\mathrm{PPL}(w_1, ..., w_n) = -\frac{\log \Pr(w_1, ..., w_n)}{n}.$$

Como pode ser observado, a perplexidade normaliza o valor da log-probabilidade pelo tamanho n da sentença. Posto de outra forma, a perplexidade calcula o custo médio do caminho que é representado pela sequência de palavras na sentença.

Como exemplo, suponha que temos as seguintes sentenças com suas respectivas probabilidades:

(1) <s> w_1 w_2 </s> [probabilidade = 0.1]
(2) <s> w_1 </s> [probabilidade = 0.4]
(3) <s> w_1 w_2 </s> [probabilidade = 0.39]

Avaliando as frases por probabilidade (ou pelo negativo da log-probabilidade), observamos que a segunda é a melhor sentença. Já ao avaliarmos por perplexidade[2], a terceira sentença se torna a melhor ($-(\log 0.4)/3 \approx 0.13 > -(\log 0.39)/4$).

Embora a perplexidade classifique as sentenças de tamanho distintos de maneira mais adequada que a simples probabilidade, ela cria um problema para os algoritmos de busca heurística. O custo de um caminho quando medido pela perplexidade não é crescente. Portanto os algoritmos de custo uniforme (ou A*) não garantem otimalidade; mais ainda, eles podem nunca terminar devido a presença de ciclos. Uma solução possível é usar algoritmos de busca que comparam em cada instante apenas sentenças de mesmo tamanho; esse é o caso tanto da busca em profundidade limitada quanto da busca em feixe. Mesmo que usemos a perplexidade, tais buscas garantidamente encontrarão uma solução (dentro do comprimento máximo dado, se existir).

Modifique sua formulação do problema e implementa a classe `CompleteSentence2` que modifica a função de custo para devolver a perplexidade de uma sentença parcial. Compare as sentenças geradas com as sentenças geradas pela sua formulação anterior.

[2]Perplexidade é uma métrica que queremos minimizar

5. Arquivos

Nós estamos disponibilizando um conjunto de arquivos para vocês executarem o trabalho. Os arquivos são:

- 5 arquivos .txt, contendo textos de diferentes naturezas que servirão como fonte para gerarmos o modelo n-grama. Não editem esses arquivos.
- utils.py contendo seis funções auxiliares. Não editem esse arquivo.
- Searches.py contendo a implementação das funções de busca. Não editem esse arquivo.
- UpperClasses.py contendo as classes LanguageModel e Problem. Não editem esse arquivo.
- completeSentence.py contendo a classe CompleteSentence, nela você deve formular e implementar o problema como um problema de busca baseado em probabilidade.
- completeSentence2.py contendo a classe CompleteSentence2, nela você deve formular e implementar o problema como um problema de busca baseado em perplexidade.
- ProblemaDeBusca.ipynb contendo explicações e juntando todos os outros arquivos. Vocês só precisarão editar duas funções nesse notebook.

Sua implementação pode ser feita em um computador local, no Google Colab ou em uma versão online do Jupyter — **tome cuidado com as versões online para não perder sua implementação**. Vocês devem entregar apenas o *completeSentence.py* e *completeSentence2.py* para avaliação.

Aviso: os textos não são curados, então os modelos podem gerar frases pesadas.