Universidade Federal do Ceará - Campus de Crateús
Bacharelado em Ciência da Computação
1º Trabalho Prático de Inteligência Artificial
22/09/2019
Prof. Lívio Freire

1 Introdução

Neste trabalho, considera-se duas tarefas: *segmentação de palavras* e *inserção de vogal*. A segmentação de palavras geralmente surge ao processar idiomas nos quais as palavras podem não ser delimitadas por espaços em ambas as extremidades, como chinês escrito ou palavras compostas longas em alemão. A inserção de vogal é relevante para idiomas como o árabe ou o hebraico, em que a escrita moderna evita notações para sons de vogais e o leitor humano os deduz do contexto. De maneira mais geral, esse é um caso de um problema de reconstrução de uma codificação, considerando algum contexto.

Já sabemos como resolver de maneira ideal qualquer problema de pesquisa específico com algoritmos de Busca em Grafos, como Busca de Custo Uniforme (BCU) ou A*. Nosso objetivo aqui é modelar - ou seja, converter tarefas do mundo real em problemas de pesquisa no espaço de estado. Dito isto, neste trabalho, sua tarefa principal não é implementar os algoritmos de busca, mas formular os problemas dessa natureza.

1.1 Modelos de Linguagem com *n-gram* e Busca de Custo Uniforme

Os algoritmos a serem implementados devem tomar decisões de segmentação e inserção com base no custo calculado de acordo com um *modelo de linguagem*. Um modelo de linguagem é uma função que captura a fluência do texto, ou seja, quantifica a facilidade de compreensão do texto.

Um modelo de linguagem comum em *Processamento de Linguagem Natural* é baseado em sequências de *n-gram*. Nessa função, dadas n palavras consecutivas, o custo é calculado com base na probabilidade (expressa em $-\log$) de que a n-ésima palavra apareça logo após as primeiras *n-1*. O custo é sempre positivo e valores baixos indicam fluência melhor. Por exemplo, no caso em que $n = 2$ e c é a função de custo *n-gram*, $c(peixe, peixe)$ deverá ser baixa, pois a probabilidade dessas duas palavras ocorrerem em sequência em português é pequena, enquanto $c(peixe, grande)$ deve ter valor maior, pois é uma construção relativamente comum. Além do mais, esses custos são aditivos.

Seja o modelo de *unigram* $u(n = 1)$. O custo atribuído à sequência $[w_1, w_2, w_3, w_4]$ é calculado da seguinte forma:

$$u(w_1) + u(w_2) + u(w_3) + u(w_4)$$

Seja o modelo de *bigram* $b(n = 1)$. O custo atribuído à sequência $[w_1, w_2, w_3, w_4]$ é calculado da seguinte forma:

$$b(w_0, w_1) + b(w_1, w_2) + b(w_2, w_3) + b(w_3, w_4)$$

onde w_0 corresponde ao *token* `-BEGIN-`, um marcador especial que denota o início de uma sentença.

As funções u e b são estimadas com base na estatística de *n-grams* calculada através de um conjunto de textos (corpus) de um certo idioma. Termos que não fazem parte do corpus recebem, pela definição da função, custos elevados.

Neste trabalho, espera-se que as implementações consigam lidar com sequências de tamanho até 200 (caracteres ou listas de itens, a depender da tarefa). Implementações capazes de lidar com sequências maiores serão bem vistas.

2 Segmentação de Palavras

Nessa tarefa, recebe-se uma sequência de caracteres ($[a - z]$) sem espaços em branco. O objetivo é inserir espaços nessa string de modo que o resultado maximize, conforme um modelo de linguagem, a fluência.

1. (2 pontos) Considere o seguinte algoritmo guloso:

   (a) inicie do início da string

   (b) encontre a posição final da próxima palavra que minimize a função de custo

   (c) repita o processo, iniciando do fim do segmento encontrado recentemente

   Mostre que esse algoritmo guloso é subótimo. Em particular, forneça um exemplo de uma string em que essa abordagem falha em obter a segmentação da entrada com o menor custo.

2. (10 pontos) Implemente um algoritmo que encontre a segmentação de palavras ótima para uma sequência de entrada. Seu algoritmo deve considerar custos baseados em uma função simples de *unigram*. Antes de implementar, é necessário estruturar essa tarefa como um problema de busca em espaços de estado. Como representar um estado? Quais os sucessores de um estado? Quais os custos de transição entre estados? Qual o teste de objetivo? A função BCU já está implementada e deve-se fazer uso dela. Para resolver o problema, preencha as funções da classe `SegmentationProblem`, especialmente o método `segmentWords`. O argumento `unigramCost` é uma função que recebe uma única string representando uma palavra e retorna o custo baseado na função *unigram*. A função `segmentWords` deverá retornar a sentença segmentada com espaços como delimitadores.

3 Inserção de Vogais

Nessa tarefa, recebe-se uma sequência de palavras em inglês sem vogais. O objetivo é inserir as vogais de volta na palavra de modo que a fluência da sentença seja maximizada (minimize o custo). Para essa tarefa, deve-se usar uma função de custo *bigram*. Além da sequência de entrada, recebe-se um mapa que produz os possíveis preenchimentos de vogais para um *token* sem vogais (`possibleFills`). Por exemplo, `possibleFills('fg')` retorna `set(['fugue', 'fog'])`.

1. (2 pontos) Considere o seguinte algoritmo guloso: da esquerda para direita, escolha repetidamente a melhor inserção imediata da vogal para a palavra sem vogal atual, dada a inserção que foi escolhida para a palavra sem vogal anterior. Esse algoritmo não leva em consideração inserções futuras além da palavra atual. Encontre um contra-exemplo realista para mostrar que esse algoritmo guloso é subótimo.

2. (10 pontos) Implemente um algoritmo para encontrar as inserções ótimas de vogais. Utilize os métodos da BCU. A função `insertVowels` deverá retornar a sequência de palavras reconstruída como uma string delimitada por espaços. Assuma que que a função recebe uma lista de strings como entrada, ou seja, que a sentença já foi quebrada em palavras anteriormente. Perceba que strings vazias são entradas válidas na lista.

   O argumento `queryWords` é a sequência de entrada das palavras sem vogal. O argumento `bigramCost` é a função que recebe duas strings representando duas palavras sequenciais e fornece o custo do *bigram*. O *token* especial `-BEGIN-` é dado por meio da constante `wordsegUtil.SENTENCE_BEGIN`. O argumento `possibleFills` é uma função que recebe uma palavra como string e retorna um conjunto de reconstruções.

   Uma vez que o corpus utilizado no exercício é limitado, algumas strings óbvias podem não ter candidato a preenchimento, como *chclt* $\rightarrow \{\}$, que tem *chocolate* como preenchimento válido. Não se preocupe sobre esses casos. Assim, se alguma palavra w sem vogal não tem candidatas a reconstrução, a implementação deve manter w mesmo sem a inserção de vogais.

4 Problema Integrado

Nessa tarefa, deve-se resolver os problemas anteriores de maneira integrada. Assim, recebe-se uma string sem espaços e vogais. O objetivo é inserir espaços e vogais nessa string visando à maximização da fluência. Como na tarefa anterior, os custos serão baseados numa função *bigram*.

1. (2 pontos) Considere o problema de busca de encontrar inserções ótimas de espaços e vogais. Formalize-o como problema de busca. Quais estados, custos, estado inicial e teste de objetivo? Busque encontrar a representação mínima de estados.

2. (20 pontos) Implemente um algoritmo para encontrar inserções ótimas de espaços e vogais. Use métodos da classe `UniformCostSearch` a partir de um objeto `ucs`. A função `segmentAndInsert` deverá retornar uma sequência de palavras reconstruídas e segmentadas como uma única string separada por espaços.

   O argumento `query` é a string de entrada. O argumento `bigramCost` é a função que recebe duas strings representando duas palavras sequenciais e fornece o custo do *bigram*.

   Diferente do problema anterior, em que palavras sem vogais podem ser consideradas uma reconstrução válida, aqui só reconstruções obtidas pela função `possibleFills` são incluídas. Além do mais, não se deve incluir palavras que contém apenas vogais (todas as palavras devem incluir pelo menos uma consoante da string de entrada).

5 Instruções

Instalação e configuração do Ambiente:

- Instalar o Miniconda: https://docs.conda.io/en/latest/miniconda.html
- Abrir o console e criar o ambiente virtual 'ia' com o Python 2.x: `conda create -n ia python=2`
- Ativar o ambiente virtual: `conda activate ia`

Todo o código relativo à solução dos problemas computacionais deve ser escrito no arquivo `submission.py`, nas regiões entre as tags `# BEGIN_YOUR_CODE` e `# END_YOUR_CODE`. A solução para as questões teóricas deve ser enviada em arquivo separado, a ser submetido no formulário da disciplina.

Para executar o código relativo às tarefas, entre no console e execute `python submission.py`. Esse programa ativará um *prompt* de comandos, que recebe as seguintes instruções:

- `seg`: segmentação de palavras
  Exemplo:
  ```
  >> seg thisisnotmybeautifulhouse
  Query (seg):  thisisnotmybeautifulhouse
  this is not my beautiful house
  ```
- `ins`: inserção de vogal
  Exemplo:
  ```
  >> ins thts m n th crnr
  Query (ins):  thts m n th crnr
  its a beautiful day in the neighborhood
  ```
- `both`: problema integrado
  Exemplo:
  ```
  >> both mgnllthppl
  Query (both):  mgnllthppl
  imagine all the people
  ```
- `fills`: consulta lista de possíveis preenchimento de vogais para um *token*
  Exemplo:
  ```
  >> fills fg
  fugue
  fog
  ```
- `ug`: retorna o custo de *unigram* de uma palavra
  Exemplo:
  ```
  >> ug fish
  11.3565088866
  ```
- `ug`: retorna o custo de *bigram* de uma sequência de duas palavras
  Exemplo:
  ```
  >> bg big fish
  13.3047699306
  ```

As soluções serão avaliadas em dois tipos de casos de teste, **basic** (básico) e **hidden** (oculto), como pode ser visto em `grader.py`. Testes básicos são completamente definidos, além de não testar os limites do código com entradas grandes e complexas, enquanto testes ocultos são mais complexos e testam os limites do código. As entradas dos testes ocultos são fornecidas em `grader.py`, mas as saídas corretas não. Para executar todos os testes, execute `python grader.py`, que irá informar quais testes básicos passaram. Para os testes ocultos, o script irá alertar se o código demora muito pra executar ou não funciona, mas não irá dizer a saída correta. Também é possível executar um teste simples (`3a-0-basic`, para o teste básico do problema integrado, por exemplo) ao executar `python grader.py 3a-0-basic`. Encoraja-se a leitura e compreensão dos casos de testes, assim com a inclusão de outros casos.

Divirta-se!