EP 1: Reconstrução de Texto via Busca

Entrega: 05/04/2020

Motivação

O objetivo deste exercício-programa é praticar a formulação de tarefas de processamento de liguagem natural como problemas de busca em espaço de estados. Em particular, lidaremos com tarefas de reconstrução de texto.

Reconstrução de texto significa recuperar um texto que foi corrompido, ou seja, algumas letras ou pontuações foram removidas. Nesse EP, vamos resolver dois tipos de reconstrução de texto: **segmentação de palavras** e **inserção de vogais**. A segmentação de palavras é usada no processamento de linguagens em que não há separação de palavras por demarcadores, como por exemplo chinês ou em palavras compostas em alemão. A inserção de vogais é usada no porcessamento de linguagens que omitem vogais, como as línguas semíticas, tais como o árabe, o hebraico e o amárico[1].

Nossa abordagem será avaliar uma reconstrução através de um **modelo de linguagem**, que é uma função que avalia a fluência de uma setença de texto. Um modelo de linguagem muito comum na área de processamento de linguagem natural é o modelo de n-gramas, e neste modelo uma possível medida de custo é dada pelo negativo do logaritmo da probabilidade empírica de ocorrência de cada palavra dadas as $(n-1)$ palavras precedentes, ou seja,

$$C(w_1, \ldots, w_m) = -\sum_{i=1}^{m} \log \Pr\Big(w_i \mid w_{i-(n-1)}, \ldots, w_{i-1}\Big)\,,$$

onde a probabilidade $\Pr(\cdot)$ é obtida pela frequência relativa a partir de um *córpus* (isto é, um conjunto de textos). Essa função é sempre positiva e quanto menor o seu valor maior a fluência da sentença. Por exemplo, se $n = 2$ dizemos que temos um modelo baseado em *bigramas*; suponha que usamos um córpus sobre vida marinha, o custo $C_b(\texttt{peixe}, \texttt{grande})$ será baixo, indicando que a frase "peixe grande" é comum, enquanto que o custo $C_b(\texttt{peixe}, \texttt{peixe})$ será alto, indicando que a frase "peixe peixe" é rara. Ademais, os custos definidos por n-gramas são aditivos; por exemplo, de acordo com um modelo de unigramas ($n = 1$), o custo atribuído a uma sentença $[w_1, w_2, w_3, w_4]$ é:

$$C_u(w_1, \ldots, w_n) = u(w_1) + u(w_2) + u(w_3) + u(w_4)\,.$$

De forma similar, o custo atribuído de acordo com um modelo de bigramas ($n = 2$) é:

$$C_b(w_1, \ldots, w_n) = b(w_0, w_1) + b(w_1, w_2) + b(w_2, w_3) + b(w_3, w_4)\,,$$

onde w_0 é `-BEGIN-`, uma palavra especial que indica o início da sentença.

[1] Googla aí para saber do que se trata!

Instruções gerais

Para esse EP você deve ter recebido um único arquivo compactado denominado `EP1-IA.zip` contendo os seguintes arquivos:

```
EP1-IA.zip
├── ep1.py
├── util.py
├── corpus.txt
├── autograder.py
└── mac0425_5739ep1.pdf
```

Você deverá implementar sua solução no esqueleto de código fornecido no arquivo `ep1.py`. Você pode criar novas funções, mas não deve modificar o protótipo das funções existentes. Cada função no esqueleto que você deve modificar está inicialmente implementada com o seguinte comando:

```
raise NotImplementedError
```

Esse comando visa informar ao programa de teste que a função não foi implementada. A primeira coisa ao escrever sua implementação para uma determinada função é remover a linha acima.

Você não precisa se preocupar em codificar um modelo de linguagem: o arquivo `util.py` contem implementações de funções que obtem u e b a partir de um córpus fornecido (`córpus.txt`). Essas funções lidam e atribuem um alto custo a palavras que não aparecem no córpus.

Desenhe suas implementações considerando que o compritmento das sequências de entrada (número de caracteres ou número de palavras, dependendo da tarefa) não ultrapassam 200. É ótimo que seu programa lide eficientemente com entradas maiores, mas isso não é esperado.

Parte 1: Segmentação de palavras

Sua primeira tarefa é implementar um algoritmo que recebe uma sequeência de caracteres (*string*) minúsculos e devolve sua segmentação em palavras com máxima fluência, usando **busca custo uniforme** com uma função de custo baseada em um modelo de unigramas ($n = 1$). Por exemplo, sua implementação deve receber `thisisnotmybeautifulhouse` e devolver `this is not my beautiful house`. Você deve *necessariamente* usar a implementação da busca de custo uniforme implementada no arquivo `util.py`,

`util.uniformCostSearch(problem)`,

e implementar sua formulação do problema de busca usando o protótipo:

```
class SegmentationProblem(util.Problem):
```

Note que essa classe deve seguir a interface descrita na classe `Problem` em `util.py`. O problema de busca deve ser resolvido usando a função:

```
def segmentWords(query, unigramCost):
```

O argumento `unigramCost` é uma função que aceita uma única string representando uma palavra e computa seu custo de acordo com um modelo de unigramas. A função `segmentWords` deve devolver a sentença segmentada com espaços como delimitadores, ou seja, usando `' '.join(words)`. Você pode supor que as strings são codificadas em ASCII.

Antes de começar a programar, procure pensar como deve ser sua formulação do problema. Por exemplo, reflita sobre como melhore representar um estado, e como é a função de transição e de custo.

Parte 2: Inserção de vogais

Sua segunda tarefa é implementar um algoritmo que recebe uma string contendo um texto (em inglês) com as vogais ausentes e devolve um novo texto contendo vogais de maneira a maximizar a fluência da frase (isto é, minimizando o custo). Você deve usar uma função de custo baseada em um modelo de bigramas e um dicionário `possibleFills` que mapeia palavras sem vogais a um conjunto de possíveis reconstruções (palavras completas). Por exemplo, a chamada `possibleFills('fg')` devolve `set(['fugue', 'fog'])`.

É você que deverá criar as funções `bigramCost` e `possibleFills` para serem usadas em `insertVowels`.

Você deve implementar sua formulação como problema de busca novamente aplicando `util.uniformCostSearch(problem)`, implementando o protótipo:

```
class VowelInsertionProblem(util.Problem):
```

O seu algoritmo de reconstrução deve ser implementado na função:

```
def insertVowels(queryWords, bigramCost, possibleFills):
```

Essa função deve devolver a sequência de palavras reconstruídas separadas por espaços, ou seja, usando `' '.join(words)`. O argumento `queryWords` é a lista de entrada de palavras sem vogais (ou seja, a sentença já foi segmentada em palavras). Note que a *string* vazia é uma palavra válida. O argumento `bigramCost` é uma função que recebe duas strings representando um duas palavras consecutivas e devolve seu custo de acordo com um modelo de bigramas. A palavra especial de início de frase com início fora do vocabulário `-BEGIN-` é dada por `util.SENTENCE_BEGIN`.

Se uma palavra não possuir reconstruções de acordo com `possibleFills`, sua implementação deve considerar a própria palavra como sua reconstrução.

Bônus de 1 ponto

Você deve ter percebido que o nosso córpus se trata de uma versão em domínio público do livro Guerra e Paz de Tolstói traduzido para o inglês. Para ganhar um ponto extra você deve adaptar o que fez nos itens 1 e 2 para o português.

Inicialmente você deverá construir um córpus em português. Sugiro que faça isso entrando no site http://dominiopublico.gov.br, selecione a obra de algum autor, por exemplo Machado de Assis, baixe toda essa obra que está no formato PDF, converta os arquivos para o formato texto, por exemplo utilizando o programa `pdftotext`, Junte todos os arquivos em um único córpus, e adapte o seu programa para poder selecionar o córpus desejado. Crie testes para o seu programa.

Insira no código um pequeno manual de uso para explicar como selecionar o córpus desejado.

Instruções para entrega

Você deve submeter via eDisciplinas apenas o arquivo `ep1.py` contendo a sua solução até às 23:55 do dia 05/04/2019. Para evitar que seu EP seja zerado, certifique-se que o arquivo foi submetido sem problemas (baixando e executando o arquivo do site) e que ele consiste em um script executável escrito em Python 3.

Avaliação

Embora seja muito importante que seu código passe em todos os testes, isso não é garantia de que seu código receberá nota máxima. É possível que os testes não verifiquem alguns casos peculiares onde seu programa pode vir a falhar.