

CODIFICAÇÃO DE ÁUDIO E VÍDEO

Modelos de Contextos Finitos

October 14, 2016

Rui Espinha Ribeiro, 68794

André Lopes, 67833

Introdução

Pretende-se verificar na prática a utilização de modelos de Markov para retirar informação estatística associada a um texto-fonte e conseguir gerar texto a partir desse mesmo texto-fonte, através de estimativas probabilísticas associadas à ocorrência de um símbolo de um alfabeto mediante um dado contexto. Este contexto é o número de símbolos que antecede o símbolo em análise. Neste trabalho, o alfabeto de símbolos em questão é o alfabeto de letras A-Z e o espaço.

Este tipo de técnica é útil para retirar ilações sobre dependências de dados, o que é particularmente útil em compressão *lossless*.

Assim, pretende-se construir uma tabela de *look-up* que mapeia diretamente todos os contextos possíveis, de uma dada ordem/profundidade e de um determinado texto-fonte, com os símbolos que se podem suceder e as suas ocorrências. A ordem indica-nos o tamanho do contexto, isto é, o número de símbolos que queremos analisar como antecessores da ocorrência de um símbolo. Com esta informação, é possível deduzir a entropia em bits associada a cada letra e a probabilidade acumulada associada ao evento "ocorrência de um símbolo precedido de um dado contexto".

Este conjunto de probabilidades acumuladas permite-nos criar, por fim, um gerador simples de texto a partir de um modelo "enriquecido" por um ou mais textos-fontes.

Implementação

Foi criada uma classe em C++ correspondente ao objecto FCM (Finite Context Model). Ao instanciar este objecto é criada uma LUT (*look-up table*) na qual é feito um mapeamento de cada contexto de ordem n encontrado no texto-fonte com os símbolos todos que ele pode preceder e o número de ocorrências de cada símbolo nesse contexto. A partir do número de ocorrências, são deduzidas as probabilidades $p(c|S)$ de ocorrência de um carácter c precedido de um contexto/estado S , que nos permitem obter a entropia associada ao modelo obtido. Conseguimos ainda obter as probabilidades acumuladas, para que através da geração de números aleatórios entre 0 e 1, seja escolhido o símbolo com a probabilidade mais próxima do número gerado relativamente a um dado contexto.

Assinatura dos métodos da classe FCM:

```
FCM(unsigned int order,          string srcText);  
typedef map<string, map<char, float> > LUT;    // look-up table datatype  
void genText(int len, int order);  
void printLUT(LUT l);  
LUT loadTable(string fileName);  
void saveTable(LUT l);  
float calcEntropy(LUT l, map<string, float> counters, float total, int order);
```

Criação da tabela de contextos

Optámos por criar uma LUT de dimensões dinâmicas: um mapa (*std::map*) que associa uma *string* de um contexto a um outro mapa, que por sua vez associa um carácter ao seu número de ocorrências. Assim, o carácter 'a' pode estar associado a 2 contagens quando, por exemplo, precedido de 'ai', mas 3 quando, por exemplo, precedido de 'ei'.

Conseguimos obter assim uma estrutura compacta e eficiente (dada a implementação interna dos mapas STL para a inserção e acesso aos seus elementos), que nos permite indexar as ocorrências dos símbolos pelos contextos, tal e qual como descrevemos anteriormente.

Levantaram-se algumas questões na construção da tabela:

- Os primeiros n caracteres do texto-fonte não têm contexto que os preceda, logo tem de haver uma maneira de garantirmos que a probabilidade destes não é zero. Optámos neste caso em utilizar os primeiros n caracteres como os primeiros caracteres do texto gerado.

- Como evitar que existam ocorrências de valor zero e consequentes probabilidades de valor zero que possam deturpar a distribuição discreta de probabilidades acumuladas? O facto de utilizarmos uma estrutura de dados dinâmica previne este caso automaticamente, pois só existirão associados a um contexto símbolos que o tenham sucedido pelo menos uma vez no texto-fonte (não é reservado espaço para todos os símbolos do nosso alfabeto em cada mapa de cada contexto).

Cálculo da entropia

Para estimar a entropia média em bits de cada letra, é necessário obter a entropia de cada estado e associá-la à probabilidade de ocorrência daquele estado. Assim, é calculada a entropia de um estado por:

$$H(S) = - \sum_{i=1}^n p(i) * \log_2 p(i) \quad (1)$$

em que n é o número de símbolos relativos a um contexto/estado S .

Para obter a entropia média de todos os estados, multiplicamos a entropia de um estado pela sua probabilidade de ocorrência:

$$H = \sum_{i=1}^m p(H(S_i)) * H(S_i) \quad (2)$$

em que m é o número de contextos que temos no nosso modelo e a $p(H(S_i))$ é dado pelo número total de ocorrências de símbolos associados a um contexto a dividir por todas as ocorrências de símbolos em toda a tabela.

Enriquecimento da tabela

Cada vez que geramos um modelo para um dado texto fonte, caso não exista previamente um ficheiro com uma LUT guardada, é criada uma de raiz que é mapeada como descrito anteriormente. Caso contrário, é carregada a LUT existente e é enriquecida com o novo texto-fonte, adicionando contextos e incrementando novos valores de ocorrências, o que nos vai permitir gerar textos diferentes a partir de tabelas mais "ricas" - ou seja, com mais contextos e com valores de ocorrências mais próximos do que é lógico dentro de uma determinada linguagem comum a vários textos-fonte.

Testes

Foram feitos vários testes com textos-modelo diferentes e com diferentes ordens para o contexto guardado, de modo a verificar quais as diferenças em termos de coesão do texto gerado e do tamanho da entropia média à medida que o valor da ordem cresce.

O executável `run.out` pode receber como argumentos os valores das flags `"-t"` (texto simples), `"-f"` (texto carregado de um ficheiro-fonte), `"-o"` (a ordem/profundidade) e `"-l"` (o comprimento do texto gerado em caracteres).

Ao ser carregado o texto do ficheiro fonte, são removidos os caracteres cujo código ASCII indique que o carácter-alvo não é uma letra do alfabeto ou um espaço.

- **Ordem 0**

Neste caso, não temos nenhum contexto, ou seja, a única variável a ter em conta é a ocorrência de um símbolo no texto todo.

```
[alopes@andrelopes assignment1]$ sudo ./run.o -t "um texto simples para mostrar ordem zero" -o 0 -l 50
Entropy: 3.63733
dr rs rioa pt mr o uteom or s a sespmozur ooa ddoe
```

Podemos verificar que em 3 execuções em separado, que os textos gerados são completamente incoerentes e complementamente díspares uns dos outros, o que pode ser explicado pela ausência de um contexto, o que retira toda a relação dos símbolos entre si.

- **Ordem $n > 0$**

Foram feitos vários testes, procurando variar o texto-fonte e a ordem, de modo a verificar a coerência do texto e o impacto sobre a entropia.

```
[alopes@andrelopes assignment1]$ sudo ./run.o -t "you will not do what i say you do what you say" -o 1 -l 60
Entropy: 2.23794
yo whay yo yo whayot whayou y will do il i yo nou dot no do n
```

No entanto, se procurarmos aumentar a ordem do contexto, obtemos resultados progressivamente mais coerentes, observando também um decréscimo da entropia ao longo das tentativas (o que é lógico, dado que quanto maior a ordem, mais determinístico é o texto gerado, pois há menos contextos e probabilidades maiores associados a estes):

```
[alopes@andrelopes assignment1]$ sudo ./run.o -t "you will not do what i say you do what you say" -o 3 -l 60
Entropy: 0.15709
you say you say you do what you will not do what you do what
[alopes@andrelopes assignment1]$ sudo ./run.o -t "you will not do what i say you do what you say" -o 4 -l 60
Entropy: 0.128527
you will not do what i say you say you do what i say you do w
[alopes@andrelopes assignment1]$ sudo ./run.o -t "you will not do what i say you do what you say" -o 5 -l 60
Entropy: 0.111618
you will not do what i say you say you do what i say you say
```

Experimentou-se também "treinar" o modelo com vários textos diferentes:

```
[alopes@andrelopes assignment1]$ sudo ./run.o -t "Ours was the marsh country, down by the river, within, as the river" -o 3 -l 60
Entropy: 0.117921
Ours was the river, within, as the river, within, as ther's
[alopes@andrelopes assignment1]$ sudo ./run.o -t "My father's family name being Pirrip, and my Christian name Philip" -o 3 -l 60
Entropy: 0.127585
My fathe marsh country, do what you within, as the river, wit
[alopes@andrelopes assignment1]$ sudo ./run.o -t "A fearful man, all in coarse gray, with a great iron on his leg. A man" -o 3 -l 60
Entropy: 0.167135
A fearful man, and my Christian name being Pirrip, and my Chr
[alopes@andrelopes assignment1]$
```

Podemos retirar uma conclusão interessante deste teste: à medida que foram dadas fontes novas ao modelo, a entropia variou em sentido ascendente, verificando que havia novas combinações possíveis para contextos existentes e contextos novos que adicionavam novas probabilidades, o que indica que é necessário um número de bits médios mais elevado para representar cada símbolo do texto aprendido.

Também testámos o nosso modelo com porções substancialmente maiores. Com um excerto com dezenas de linhas do livro *The Picture of Dorian Gray*, de Oscar Wilde (excerto que segue em anexo) verificamos uma entropia média baixíssima (com ordem $n=4$ e 600 carateres de texto gerado):

```
Entropy: 0.572147
As they neithere had so many fascinates me, but whose name your best the was we harmony of any resemblance he moment on the know not being at the becomes all physical and thin his brai
bette. Not send it anywhere is a natural consequence becomes all should be sough the feared they neithere.Lord Henry stroys their ease answered the been able of course I am teltectual
et. The ugly and closing at least like him.You don't seems too vulgar. Whenever terribly.
```

Associada à baixa entropia, está naturalmente um texto muito mais "coerente", na relação entre as suas palavras.

Discussão

Conclusões

Consegue-se verificar, após vários testes, que quanto maior é a ordem dada menor é a entropia calculada, o que faz sentido tendo em conta que quanto maior é a ordem menores são as variações verificadas no texto gerado, dado haver um maior contexto e, consequentemente, um menor número de combinações possíveis, cujas probabilidades associadas são muito superiores.

Foi possível também verificar que quanto mais combinações forem dadas entre palavras, maior é a entropia e consequentemente o texto gerado é cada vez mais diverso.

No entanto, pudemos evidenciar um caso em que, com uma porção de texto significativa extraída de um livro, pudemos verificar uma baixa entropia, o que indica que o texto gerado a partir da porção-fonte será mais determinístico.

Possíveis melhorias

O ficheiro de execução e teste da classe FCM contém uma função "normalizadora", que remova todos os caracteres que não sejam, espaços ou letras do alfabeto (maiúsculas ou minúsculas). Não incluímos caracteres "particulares", com acentos ou semelhante pois revelou-se já bastante fora do nosso objetivo o trabalho a desenvolver neste âmbito. Tinha sido criado um dicionário para caracteres invulgares ("ç", "á", etc). No entanto, verificámos que cada carater alterado se dividia em dois (um deles o normalizado) aquando desta conversão, devido a haver mais de um byte de *encoding* neste tipo de caracteres. Caso esta conversão estivesse completamente implementada, poderíamos ler texto de vários idiomas, pois ele seria normalizado, para que estivesse sempre a analisar os resultados com um alfabeto comum.