Trabalho de Conclusão da Disciplina Processamento de Linguagem Natural

Vítor Carneiro Curado

19 de novembro de 2018

Sumário

1 Introdução

O presente trabalho foi desenvolvido como requisito para conclusão da disciplina Processamento de Linguagem Natural, do programa de pós-graduação *Lato Sensu* em Informática, área de concentração em Gestão de Tecnologia da Informação. O objetivo do trabalho é realizar uma avaliação intrínseca de modelos de linguagem neuronais.

Para isso, foi utilizada a ferramenta *word2vec*, que fornece uma implementação das arquiteturas *continuous bag-of-words (CBOW)* and *skip-gram*. A ferramenta em questão está disponível no endereço: <https://github.com/tmikolov/word2vec>.

De acordo com as instruções do trabalho prático, a avaliação dos modelos de linguagem deverá variar o tamanho do *corpus*, o tamanho de contexto e, também, a arquitetura CBOW e Skip-Gram.

Considerando o exposto, a seção 2 apresenta a metodologia utilizada na realização do trabalho. Por sua vez, a seção 3 apresenta a avaliação realizada dos modelos. Por fim, a seção 4 conclui o trabalho.

2 Metodologia

A fim de avaliar diferentes modelos de linguagem, foram desenvolvidos dois programas: um *bash script* para criação dos *wordspaces*[1] e um programa em C similar ao *word-analogy*.

O *bash script* teve a finalidade de criar diferentes *wordspaces* variando de forma automatizada o tamanho de contexto, a arquitetura (CBOW e Skip-Gram) e o tamanho do *corpus* utilizado para treinar o modelo. O script em questão está no apêndice A.

O programa em C, por sua vez, teve o propósito de avaliar o *wordspace* criado. Essa avaliação foi realizada utilizando o arquivo *questions-words.txt*. Esse arquivo contem 19.544 linhas de texto, sendo que cada linha contem 4 palavras. As 3 primeiras palavras são utilizadas pelo programa para tentar calcular o vetor da 4ª palavra, sendo que a 4ª palavra que consta no arquivo *questions-words.txt* é o resultado correto. O erro do modelo, portanto, é calculado com base na distância euclidiana entre a palavra calculada pelo programa e a palavra correta, que consta no arquivo. O código-fonte do programa criado está disponível no apêndice B.

Deste modo, foram avaliados *wordspaces* CBOW e Skip-Gram, com tamanho de contexto variando de dois em dois, sendo que o menor contexto avaliado foi de tamanho 2 e o maior foi de tamanho 20. Por sua vez, para treinar o modelo foram utilizados 10 (dez) diferentes tamanhos de *corpus*. Os tamanhos de *corpus* utilizados foram (em quantidade de palavras): 1.650.000, 3.300.000, 5.100.000, 6.750.000, 8.550.000, 10.200.000, 12.000.000, 13.650.000, 15.300.000 e 17.005.207.

[1] *Wordspace* é um espaço de vetores de palavras.

Todos os códigos e arquivos utilizados na realização deste trabalho estão disponíveis para download no endereço:

<https://github.com/carneirocurado/Processamento-Linguagem-Natural>

3 Avaliação dos Modelos

A avaliação foi realizada considerando todas as 19.544 linhas de texto do arquivo *questions-words.txt*. Para cada linha do arquivo era verificado, em primeiro lugar, se todas as quatro palavras estavam presentes no modelo. Caso alguma dessas quatro palavras não existisse no modelo, essa linha era classificada como *Fora do Dicionário*. Caso, entretanto, todas as quatro palavras estivessem presentes no modelo, os vetores das três primeiras palavras eram utilizados em uma operação algébrica visando encontrar um vetor resultante que tivesse uma relação com o vetor da terceira palavra similar à relação que o vetor da segunda palavra tem com o da primeira. A equação 1 exemplifica essa operação para o caso de a linha do arquivo ser: *Athens Greece Paris France*.

$$vetor\_resultante = \left(\overbrace{vetor_2}^{Greece} - \overbrace{vetor_1}^{Athens} \right) + \overbrace{vetor_3}^{Paris} \tag{1}$$

Após realizado o cálculo da equação 1, é esperado que o $vetor\_resultante$ corresponda à palavra *France*, isto é, a palavra *France* deve ser, dentre todas as outras palavras do vocabulário, a mais próxima ao $vetor\_resultante$. Para, então, verificar a acurácia do modelo, é calculada a distância euclidiana entre o $vetor\_resultante$ e todas as outras palavras do vocabulário. Um *ranking*, então, é montado com as 1.000 (mil) palavras mais próximas do $vetor\_resultante$. Por fim, é verificada qual a posição da palavra correta (*France*) nesse *ranking*. Caso a palavra *France* esteja fora desse *ranking*, ela é contabilizada como *Fora do Ranking*. Isto é, fora do *ranking* são as palavras que estão no dicionário do modelo mas não estão entre as 1.000 palavras mais próximas do $vetor\_resultante$. Caso, entretanto, a palavra *France* esteja no *ranking*, o erro do modelo é calculado com base na equação 2, que calcula a raiz do erro quadrático médio (*Root Mean Squared Deviation* - RMSD) do modelo.

$$RMSD = \sqrt{\frac{\sum_{i=1}^{qtde} \left(d(x) - d(0)\right)^2}{qtde}} \tag{2}$$

onde:

$qtde$: total de linhas analisadas
$d(x)$: distância euclidiana do vetor_resultante à palavra *France*
$d(0)$: distância euclidiana do vetor_resultante à palavra mais próxima a ele

Conforme pode ser observado pela equação 2, caso a palavra *France* seja a mais próxima ao *vetor_resultante* o erro será zero. O RMSD final do modelo é a raiz do somatório do quadrado do erro de cada linha contabilizada na análise, dividido pelo total de linhas contabilizadas. As linhas contabilizadas são o total de linhas do arquivo *question-words.txt* descontadas as linhas com palavras fora do dicionário ou fora do *ranking*. Cabe destacar que quanto maior o *corpus* utilizado, menor era a quantidade de linhas fora do dicionário. A figura 5 apresenta essa relação. Por sua vez, a quantidade de linhas fora do *ranking* era influenciada pelo tamanho do *corpus*, pelo tamanho do contexto e pela arquitetura do modelo (CBOW ou Skip-Gram). A figura 1 ilustra essa relação para a arquitetura CBOW e a figura 2 para a Skip-Gram.

Tamanho do	Tamanho do Contexto CBOW									
Corpus (palavras)	2	4	6	8	10	12	14	16	18	20
1.650.000	10,37%	8,18%	8,08%	7,71%	7,87%	8,06%	7,80%	7,94%	8,38%	8,25%
3.300.000	12,17%	8,69%	8,22%	7,79%	7,80%	8,38%	8,15%	8,60%	8,45%	8,88%
5.100.000	13,14%	8,82%	8,51%	7,58%	8,29%	8,40%	8,48%	8,07%	8,67%	9,17%
6.750.000	11,82%	7,42%	6,81%	6,59%	6,75%	6,87%	7,19%	7,69%	7,57%	8,05%
8.550.000	11,47%	6,87%	6,29%	6,12%	6,40%	6,45%	6,47%	6,34%	7,20%	7,22%
10.200.000	10,34%	6,23%	5,74%	5,71%	6,15%	6,12%	6,43%	5,78%	6,78%	6,78%
12.000.000	10,61%	6,70%	6,04%	5,71%	6,34%	6,37%	6,52%	6,85%	6,74%	6,98%
13.650.000	9,72%	6,20%	5,67%	5,25%	5,74%	5,60%	6,31%	6,27%	6,00%	6,60%
15.300.000	10,32%	6,64%	6,12%	5,99%	6,11%	6,46%	6,24%	6,44%	6,84%	6,77%
17.005.207	10,32%	6,33%	5,97%	6,11%	6,25%	6,12%	6,32%	6,52%	6,39%	6,75%

Figura 1 – Percentual de linhas fora do *ranking* usando o modelo CBOW variando o tamanho do Corpus e o tamanho do Contexto

Tamanho do	Tamanho do Contexto Skip-Gram									
Corpus (palavras)	2	4	6	8	10	12	14	16	18	20
1.650.000	13,15%	12,28%	10,33%	10,07%	9,68%	9,80%	9,61%	9,29%	9,44%	9,29%
3.300.000	15,53%	13,51%	10,55%	9,74%	9,62%	9,09%	8,93%	8,90%	9,31%	9,06%
5.100.000	16,59%	13,26%	10,36%	9,15%	9,00%	8,86%	8,35%	8,46%	8,26%	8,43%
6.750.000	15,06%	10,80%	7,79%	7,19%	6,83%	6,27%	6,39%	6,40%	6,33%	6,86%
8.550.000	14,75%	10,44%	8,01%	7,93%	7,09%	7,20%	7,24%	6,93%	7,07%	7,47%
10.200.000	12,08%	8,93%	7,22%	6,74%	6,67%	6,40%	6,44%	6,60%	6,82%	6,33%
12.000.000	13,24%	9,72%	7,53%	7,27%	7,12%	7,22%	7,01%	7,17%	7,41%	7,13%
13.650.000	11,15%	8,78%	7,00%	6,66%	6,76%	6,39%	6,49%	6,31%	6,69%	6,60%
15.300.000	12,36%	9,30%	7,21%	6,39%	6,47%	6,49%	6,67%	6,45%	7,00%	6,90%
17.005.207	11,34%	8,76%	6,26%	6,22%	6,34%	6,05%	6,07%	6,05%	6,33%	6,25%

Figura 2 – Percentual de linhas fora do *ranking* usando o modelo Skip-gram variando o tamanho do Corpus e o tamanho do Contexto

Conforme pode ser observado nas figuras 1 e 2, aumentar o tamanho do *corpus* sem levar em consideração outros fatores pode levar ao aumento no percentual de linhas classificadas como fora do *ranking*. Para que a análise seja completa, o erro do modelo deve ser considerado. A figura 3 apresenta a raiz do erro quadrático médio (*Root Mean Squared Deviation* - RMSD) do modelo CBOW e a figura 4 apresenta o RMSD para o modelo Skip-Gram. Observando as figuras 3 e 4 é possível notar que o aumento do tamanho do *corpus* sempre diminui o RMSD dos modelos.

A análise da figura 3 permite observar que o RMSD mínimo para o modelo CBOW foi de $0,076045$ com um contexto de tamanho 14. Por sua vez, o RMSD mínimo para o

Tamanho do Corpus (palavras)	Tamanho do Contexto CBOW									
	2	4	6	8	10	12	14	16	18	20
1.650.000	0,125469	0,119894	0,120972	0,122458	0,123663	0,123389	0,122333	0,126605	0,125709	0,125686
3.300.000	0,105949	0,102941	0,101806	0,10117	0,099327	0,102868	0,101805	0,103924	0,103122	0,103499
5.100.000	0,10172	0,096583	0,09378	0,095087	0,093231	0,093253	0,093512	0,094958	0,093767	0,094208
6.750.000	0,096644	0,092339	0,091163	0,090449	0,088556	0,090126	0,089159	0,091099	0,090314	0,090037
8.550.000	0,094887	0,089619	0,087811	0,087802	0,088056	0,087636	0,088983	0,086966	0,087485	0,087246
10.200.000	0,090554	0,085456	0,08312	0,083997	0,083414	0,083836	0,085087	0,083232	0,083236	0,086771
12.000.000	0,089713	0,082977	0,081922	0,081369	0,082401	0,081044	0,081888	0,082111	0,081951	0,081237
13.650.000	0,085887	0,080916	0,080311	0,079568	0,079966	0,081745	0,079578	0,082285	0,08208	0,080702
15.300.000	0,085092	0,078795	0,077629	0,078847	0,07874	0,078795	0,077145	0,078812	0,078189	0,079627
17.005.207	0,085029	0,078387	0,076546	0,076072	0,076653	0,076059	0,076045	0,077304	0,077958	0,078795

Figura 3 – *Root Mean Squared Deviation* do modelo CBOW variando o tamanho do Corpus e o tamanho do Contexto

Tamanho do Corpus (palavras)	Tamanho do Contexto Skip-Gram									
	2	4	6	8	10	12	14	16	18	20
1.650.000	0,096652	0,101837	0,103403	0,103679	0,103089	0,102846	0,104039	0,104375	0,102765	0,104508
3.300.000	0,085841	0,087233	0,089096	0,08946	0,091733	0,09219	0,090952	0,093218	0,091854	0,093242
5.100.000	0,081157	0,082936	0,082174	0,081878	0,082827	0,085264	0,084653	0,084824	0,08528	0,085294
6.750.000	0,078004	0,079481	0,077154	0,0787	0,079235	0,080455	0,081255	0,080755	0,080728	0,081223
8.550.000	0,074072	0,07518	0,075384	0,075368	0,075902	0,077542	0,078722	0,078779	0,079014	0,080134
10.200.000	0,071635	0,071934	0,07095	0,071438	0,073678	0,072848	0,073961	0,074892	0,075609	0,075878
12.000.000	0,070146	0,071186	0,068741	0,070635	0,071433	0,071314	0,070726	0,07214	0,075162	0,073164
13.650.000	0,068063	0,068683	0,066422	0,067647	0,069514	0,069799	0,068631	0,07032	0,070966	0,071106
15.300.000	0,066029	0,064645	0,065607	0,067553	0,066772	0,067628	0,06945	0,069037	0,071047	0,071428
17.005.207	0,066538	0,065966	0,064336	0,065215	0,065642	0,067292	0,066484	0,06806	0,067513	0,06957

Figura 4 – *Root Mean Squared Deviation* do modelo Skip-gram variando o tamanho do Corpus e o tamanho do Contexto

modelo Skip-Gram foi de $0,064336$ com um contexto de tamanho 6. A figura 5 apresenta o efeito da variação do tamanho do *corpus* com o tamanho de contexto fixo nos melhores valores encontrados para o RMSD.

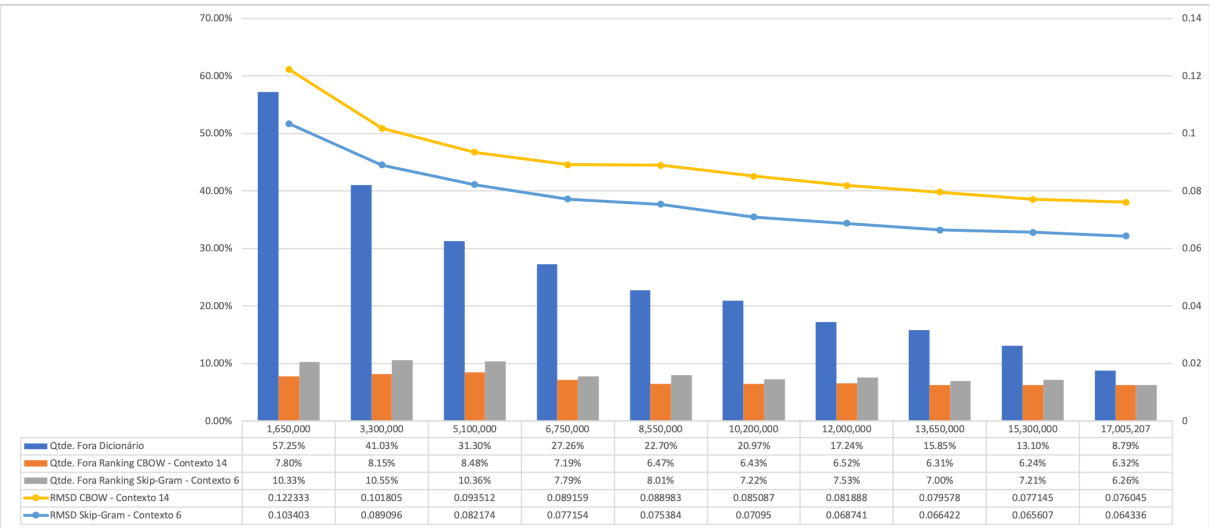

Figura 5 – Efeito da variação do tamanho do *corpus* com o tamanho de contexto fixo nos melhores valores encontrados

4 Conclusão

O presente trabalho realizou uma avaliação de diferentes modelos de linguagem variando o tamanho do *corpus* utilizado para treinar o modelo, o tamanho de contexto e, também, a arquitetura CBOW e Skip-Gram.

A fim de realizar essa avaliação, foram desenvolvidos dois programas: um *bash script* para criação dos *wordspaces* e um programa em C similar ao *word-analogy*[2].

O *bash script* teve a finalidade de criar diferentes *wordspaces* variando de forma automatizada o tamanho de contexto, a arquitetura (CBOW e Skip-Gram) e o tamanho do *corpus* utilizado para treinar o modelo. O programa em C, por sua vez, teve o propósito de avaliar o *wordspace* criado.

Conforme detalhado na seção 3, quanto maior era o tamanho do *corpus* utilizado, menor era o percentual de palavras fora do dicionário do modelo. Por sua vez, o percentual de palavras fora do *ranking* era afetado pelo tamanho do *corpus*, pelo tamanho do contexto e pela arquitetura do modelo (CBOW ou Skip-Gram).

Para toda variação de parâmetro foi calculada a raiz do erro quadrático médio (*Root Mean Squared Deviation* - RMSD) do modelo. O menor RMSD para a arquitetura CBOW foi com um contexto de tamanho 14 e, para a arquitetura Skip-Gram, o menor RMSD foi com um contexto de tamanho 6. Esses resultados estão detalhados na seção 3. A figura 5 ilustra o efeito da variação do tamanho do *corpus* com o tamanho de contexto fixo nesses melhores valores encontrados. Conforme pode ser observado, a arquitetura Skip-Gram foi ligeiramente superior que a CBOW pois, além de apresentar menor erro (RMSD), também apresentou um menor índice de palavras fora do *ranking*. Os modelos que adotavam a arquitetura Skip-Gram, entretanto, foram significativamente mais lentos de se construir que os que adotavam o CBOW.

[2] Todos os códigos e arquivos utilizados na realização deste trabalho estão disponíveis para download no endereço: <https://github.com/carneirocurado/Processamento-Linguagem-Natural>

APÊNDICE A – Script criação dos Wordspaces

```
1  #!/bin/bash
2
3  ARQ_IN=$1
4  ARQ_SAIDA=$2
5
6  timestamp(){
7          date +"%Y-%m-%d %T"
8  }
9
10 printf "\n%s: Iniciando Execucao..." "$(timestamp)"
11
12 QTDE_FILES=$(ls -1 ./corpus2 | wc -l)
13 qtde_processada=0
14
15 for arquivo in ./corpus2/*; do
16         qtde_processada=$(($qtde_processada+1))
17         printf "\n----------\n%s: Corpus de
               Entrada: %s - %d de %d" "$(timestamp)" "$(basename
               $arquivo)" "$qtde_processada" "$QTDE_FILES"
18
19         ARQ_EXISTE=$(find ./wordspaces -name *$(basename
               $arquivo)* | wc -w)
20
21 #       TEMP="text8_12600000"
22 #       AUX=$(basename $arquivo)
23 #       if [ "$AUX" = "$TEMP" ]
24
25         if [ $ARQ_EXISTE = 0 ]
26         then
27                 ARQ_SIZE=$(echo $(basename $arquivo) | cut -d _
                       -f 2)
28                 ARQ_SIZE_REAL=$(more $arquivo | wc -w)
29                 printf "\nQuantidade esperada de palavras: %d -
                       Quantidade Encontrada: %d" "$ARQ_SIZE" "
                       $ARQ_SIZE_REAL"
30                 if [ $ARQ_SIZE -ne $ARQ_SIZE_REAL ]
```

```
31                      then
32                              printf "\nWARNING: Quantidade encontrada
                                    de palavras diferente da esperada!
                                    Pulando..."
33                      else
34                              for ((i=2; i<=20; i=$(($i+2)))); do
35  #                                   if [ $i -gt 20 ]
36  #                                   then
37  #                                           i=$(($i+4))
38  #                                           printf "\n\n%s: Termino
        da Execucao\n" "$(timestamp)"
39  #                                           exit 0
40  #                                   fi
41                                      ARQ_WORDSPACE=./wordspaces/$(
                                            basename $arquivo)
                                            _cbow_context-$i.bin
42
43                                      printf "\n
                                            ------------------------------------\
                                            n%s: Iniciando Construcao do
                                            Wordspace: %s\n" "$(timestamp
                                            )" "$ARQ_WORDSPACE"
44                                      time ./word2vec/word2vec -train
                                            $arquivo -output
                                            $ARQ_WORDSPACE -cbow 1 -size
                                            200 -window $i -negative 25 -
                                            threads 20 -binary 1 -iter 15
45                                      printf "\n\n%s: Iniciando
                                            Analise: %s\n" "$(timestamp)"
                                             "$ARQ_WORDSPACE"
46                                      ./word-analogy2 $ARQ_WORDSPACE
                                            $ARQ_SAIDA < $ARQ_IN
47
48                                      ARQ_WORDSPACE=./wordspaces/$(
                                            basename $arquivo)_skip-
                                            gram_context-$i.bin
49                                      printf "\n
                                            ------------------------------------\
                                            n%s: Iniciando Construcao do
```

```
                                        Wordspace: %s\n" "$(timestamp
                                        )" "$ARQ_WORDSPACE"
50                                      time ./word2vec/word2vec -train
                                        $arquivo -output
                                        $ARQ_WORDSPACE -cbow 0 -size
                                        200 -window $i -negative 25 -
                                        threads 20 -binary 1 -iter 15
51                                      printf "\n\n%s: Iniciando
                                        Analise: %s\n" "$(timestamp)"
                                         "$ARQ_WORDSPACE"
52                                      ./word-analogy2 $ARQ_WORDSPACE
                                        $ARQ_SAIDA < $ARQ_IN
53                              done
54                      fi
55              else
56                      printf "\nWARNING: Corpus %s ja processado!
                           Pulando...\n" "$(basename $arquivo)"
57              fi
58  done
59
60  printf "\n\n%s: Termino da Execucao\n" "$(timestamp)"
```

APÊNDICE B – Código-fonte

```
1  //
2  //  Licensed under the Apache License, Version 2.0 (the "License
       ");
3  //  you may not use this file except in compliance with the
       License.
4  //  You may obtain a copy of the License at
5  //
6  //       http://www.apache.org/licenses/LICENSE-2.0
7  //
8  //  Unless required by applicable law or agreed to in writing,
       software
9  //  distributed under the License is distributed on an "AS IS"
       BASIS,
10 //  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express
```

```
//  or implied.
//  See the License for the specific language governing permissions and
//  limitations under the License.

#include <stdio.h>
#include <string.h>
#include <math.h>
#include <stdlib.h>
#include <ctype.h>
#include <time.h>

const long long max_size = 2000;         // max length of strings
const long long N = 1000;                // number of closest words that will be shown
const long long max_w = 50;              // max length of vocabulary entries
const long long max_cat = 100;           // quantidade maxima de categorias de palavras que serao tratadas

int read_words(char st[100][max_size]);
int find_words (char st[100][max_size], long long *bi, char *vocab, long long words);
int find_nearest (long long words, float *vec, char *vocab, float *M, long long *bi, long long size, float *bestd, char bestw[N][max_size]);

int main(int argc, char **argv) {
  FILE *f, *arq_out;
  time_t hora, hora_aux;
  struct tm *hora_local;
//  char st1[max_size];
  char bestw[N][max_size], cat_name[max_cat][max_w];
  char wordspace_file[max_size], arq_saida_file[max_size], st[100][max_size];
  float erro, mse, rmsd, cat_mse[max_cat], cat_rmsd[max_cat], len, bestd[N], vec[max_size];
  long long words, size, a, b, cn, bi[100], qtde_out_dictionary,
```

```
        qtde_out_score, cat_qtde_out_dictionary[max_cat],
        cat_qtde_out_score[max_cat], cat_qtde_total[max_cat],
        qtde_total, qtde_cat;
39    float *M;
40    char *vocab;
41    if (argc < 3) {
42      printf("Utilizacao: ./word-analogy2 <WORDSPACE> <ARQ_SAIDA>
          < <ARQ_ENTRADA>\nonde:\n\tWORDSPACE contem projecoes de
          palavras no formato binario\n\tARQ_SAIDA indica o nome do
          arquivo de saida para armazenar o resultado do
          processamento\n\t< ARQ_ENTRADA indica o redirecionamento
          da entrada padrao para um arquivo com dados a serem
          processados.");
43      return 0;
44    }
45    strcpy(wordspace_file, argv[1]);
46    f = fopen(wordspace_file, "rb");
47    if (f == NULL) {
48      printf("Arquivo %s nao encontrado.\n", argv[1]);
49      return -1;
50    }
51    strcpy(arq_saida_file, argv[2]);
52    arq_out = fopen(arq_saida_file, "a");
53    if (f == NULL) {
54      printf("Arquivo %s nao encontrado.\n", argv[2]);
55      return -1;
56    }
57    fscanf(f, "%lld", &words);
58    fscanf(f, "%lld", &size);
59    vocab = (char *)malloc((long long)words * max_w * sizeof(char)
        );
60    M = (float *)malloc((long long)words * (long long)size *
        sizeof(float));
61    if (M == NULL) {
62      printf("Cannot allocate memory: %lld MB    %lld  %lld\n", (
          long long)words * size * sizeof(float) / 1048576, words,
          size);
63      return -1;
64    }
```

```
65    for (b = 0; b < words; b++) {
66      a = 0;
67      while (1) {
68        vocab[b * max_w + a] = fgetc(f);
69        if (feof(f) || (vocab[b * max_w + a] == ' ')) break;
70        if ((a < max_w) && (vocab[b * max_w + a] != '\n')) a++;
71      }
72      vocab[b * max_w + a] = 0;
73      for (a = 0; a < max_w; a++) vocab[b * max_w + a] = toupper(
          vocab[b * max_w + a]);
74      for (a = 0; a < size; a++) fread(&M[a + b * size], sizeof(
          float), 1, f);
75      len = 0;
76      for (a = 0; a < size; a++) len += M[a + b * size] * M[a + b
          * size];
77      len = sqrt(len);
78      for (a = 0; a < size; a++) M[a + b * size] /= len;
79    }
80    fclose(f);
81
82    mse = 0;
83    qtde_out_dictionary = 0;
84    qtde_out_score = 0;
85    qtde_total = 0;
86    qtde_cat = -1;
87    hora = time(NULL);
88    if ( hora == (time_t)-1 ) printf ("WARNING: Erro resgatando
        hora local\n");
89    hora_local = localtime(&hora);
90    printf ("%d-%02d-%02d %02d:%02d:%02d - Iniciando Processamento
        \n", hora_local->tm_year+1900, hora_local->tm_mon+1,
        hora_local->tm_mday, hora_local->tm_hour, hora_local->
        tm_min, hora_local->tm_sec);
91    if ( ftell(arq_out) == 0 )
92      if ( (fprintf(arq_out, "wordspace;categoria;qtde_total;
          qtde_analisada;qtde_fora_dicionario;qtde_fora_ranking;mse
          ;rmsd\n")) < 0 ) {
93        printf ("\nERROR: Falha escrevendo no arquivo de Saida\n")
            ;
```

```
94        return -1;
95      }
96    while (1) {
97      cn = read_words(st);
98      if ( cn < 0 ) {
99        if ( qtde_cat >= 0 ) {
100         // Calculando o MSE e o RMSD do processamento da
              categoria anterior
101         if ( cat_qtde_total[qtde_cat] > 0 ) {
102           cat_mse[qtde_cat] /= cat_qtde_total[qtde_cat];
103           cat_rmsd[qtde_cat] = sqrtf(cat_mse[qtde_cat]);
104         }
105         else {
106           cat_mse[qtde_cat] = 0;
107           cat_rmsd[qtde_cat] = 0;
108         }
109         printf ("\rTermino da Categoria: %s - Quantidade Total: 
              %lld - Analisada: %lld (%lld fora do dicionario e %
              lld fora do ranking) - MSE: %f - RMSD: %f          \n
              ", cat_name[qtde_cat], cat_qtde_total[qtde_cat]+
              cat_qtde_out_dictionary[qtde_cat]+cat_qtde_out_score[
              qtde_cat], cat_qtde_total[qtde_cat],
              cat_qtde_out_dictionary[qtde_cat], cat_qtde_out_score
              [qtde_cat], cat_mse[qtde_cat], cat_rmsd[qtde_cat]);
110
111         if ( (fprintf(arq_out, "%s;%s;%lld;%lld;%lld;%lld;%f;%f\
              n", wordspace_file, cat_name[qtde_cat],
              cat_qtde_total[qtde_cat]+cat_qtde_out_dictionary[
              qtde_cat]+cat_qtde_out_score[qtde_cat],
              cat_qtde_total[qtde_cat], cat_qtde_out_dictionary[
              qtde_cat], cat_qtde_out_score[qtde_cat], cat_mse[
              qtde_cat], cat_rmsd[qtde_cat])) < 0 ) {
112           printf ("\nERROR: Falha escrevendo no arquivo de Saida
                \n");
113           return -1;
114         }
115       }
116       break;
117     }
```

```
118     if ( !cn ) {
119       if ( qtde_cat >= 0 ) {
120         // Calculando o MSE e o RMSD do processamento da
              categoria anterior
121         if ( cat_qtde_total[qtde_cat] > 0 ) {
122           cat_mse[qtde_cat] /= cat_qtde_total[qtde_cat];
123           cat_rmsd[qtde_cat] = sqrtf(cat_mse[qtde_cat]);
124         }
125         else {
126           cat_mse[qtde_cat] = 0;
127           cat_rmsd[qtde_cat] = 0;
128         }
129         printf ("\rTermino da Categoria: %s - Quantidade Total:
              %lld - Analisada: %lld (%lld fora do dicionario e %
              lld fora do ranking) - MSE: %f - RMSD: %f          \n
              ", cat_name[qtde_cat], cat_qtde_total[qtde_cat]+
              cat_qtde_out_dictionary[qtde_cat]+cat_qtde_out_score[
              qtde_cat], cat_qtde_total[qtde_cat],
              cat_qtde_out_dictionary[qtde_cat], cat_qtde_out_score
              [qtde_cat], cat_mse[qtde_cat], cat_rmsd[qtde_cat]);
130
131         if ( (fprintf(arq_out, "%s;%s;%lld;%lld;%lld;%lld;%f;%f\
              n", wordspace_file, cat_name[qtde_cat],
              cat_qtde_total[qtde_cat]+cat_qtde_out_dictionary[
              qtde_cat]+cat_qtde_out_score[qtde_cat],
              cat_qtde_total[qtde_cat], cat_qtde_out_dictionary[
              qtde_cat], cat_qtde_out_score[qtde_cat], cat_mse[
              qtde_cat], cat_rmsd[qtde_cat])) < 0 ) {
132           printf ("\nERROR: Falha escrevendo no arquivo de Saida
                \n");
133           return -1;
134         }
135       }
136       qtde_cat++;
137       if ( qtde_cat > max_cat ) {
138         qtde_cat = max_cat;
139         printf ("\nWARNING: Quantidade maxima de categorias
              estourada!\n");
140       }
```

```
141
142         hora = time(NULL);
143         if ( hora == (time_t)-1 ) printf ("\nWARNING: Erro
                resgatando hora local\n");
144         hora_local = localtime(&hora);
145         printf ("----------\n%d-%02d-%02d %02d
                :%02d:%02d - Iniciando Categoria: %s\n", hora_local->
                tm_year+1900, hora_local->tm_mon+1, hora_local->tm_mday
                , hora_local->tm_hour, hora_local->tm_min, hora_local->
                tm_sec, st[0]);
146
147         strncpy(cat_name[qtde_cat],st[0],max_w-1);
148         cat_name[qtde_cat][max_w-1] = '\0';
149         cat_mse[qtde_cat] = 0;
150         cat_rmsd[qtde_cat] = 0;
151         cat_qtde_total[qtde_cat] = 0;
152         cat_qtde_out_score[qtde_cat] = 0;
153         cat_qtde_out_dictionary[qtde_cat] = 0;
154         continue;
155       }
156
157       qtde_total += 1;
158       cat_qtde_total[qtde_cat]++;
159
160       hora_aux = time(NULL);
161       if ( hora_aux == (time_t)-1 ) printf ("\nWARNING: Erro
              resgatando hora local\n");
162       hora_local = localtime(&hora_aux);
163       printf ("\r%d-%02d-%02d %02d:%02d:%02d - Quantidade Total: %
              lld - Analisada: %lld (%lld fora do dicionario e %lld
              fora do ranking) - tempo de processamento na categoria %f
              seg", hora_local->tm_year+1900, hora_local->tm_mon+1,
              hora_local->tm_mday, hora_local->tm_hour, hora_local->
              tm_min, hora_local->tm_sec, cat_qtde_total[qtde_cat]+
              cat_qtde_out_dictionary[qtde_cat]+cat_qtde_out_score[
              qtde_cat], cat_qtde_total[qtde_cat],
              cat_qtde_out_dictionary[qtde_cat], cat_qtde_out_score[
              qtde_cat], difftime(hora_aux,hora));
164       fflush(stdout);
```

```
165
166
167       if ( find_words(st, bi, vocab, words) <= 0 ) {
168         qtde_out_dictionary++;
169         cat_qtde_out_dictionary[qtde_cat]++;
170         qtde_total--;
171         cat_qtde_total[qtde_cat]--;
172         continue;
173       }
174
175       find_nearest (words, vec, vocab, M, bi, size, bestd, bestw);
176
177       erro = -1;
178       for (a = 0; a < N; a++) {
179         if (!strcmp(bestw[a], st[3])) {
180           erro = powf(bestd[a] - bestd[0],2);
181           mse += erro;
182           cat_mse[qtde_cat] += erro;
183           break;
184         }
185       }
186       if ( erro < 0 ) {
187         qtde_out_score++;
188         cat_qtde_out_score[qtde_cat]++;
189         qtde_total--;
190         cat_qtde_total[qtde_cat]--;
191       }
192     }
193
194     if ( qtde_total > 0 ) {
195       mse /= qtde_total;
196       rmsd = sqrtf(mse);
197     }
198     else {
199       mse = 0;
200       rmsd = 0;
201     }
202
203     printf ("--------------------------------\n%d-%02d-%02d %02d:%02d
```

```
        :%02d - Termino da Execucao - Quantidade Total: %lld -
        Analisada: %lld (%lld fora do dicionario e %lld fora do
        ranking) - MSE: %f - RMSD: %f\n
        ----------\n", hora_local->tm_year+1900,
         hora_local->tm_mon+1, hora_local->tm_mday, hora_local->
        tm_hour, hora_local->tm_min, hora_local->tm_sec, qtde_total
        +qtde_out_score+qtde_out_dictionary, qtde_total,
        qtde_out_dictionary, qtde_out_score, mse, rmsd);
204
205     if ( (fprintf(arq_out, "%s;TOTAL;%lld;%lld;%lld;%lld;%f;%f\n",
             wordspace_file, qtde_total+qtde_out_score+
        qtde_out_dictionary, qtde_total, qtde_out_dictionary,
        qtde_out_score, mse, rmsd)) < 0 ) {
206       printf ("\nERROR: Falha escrevendo no arquivo de Saida\n");
207       return -1;
208     }
209
210     fclose(arq_out);
211
212     return 0;
213   }
214
215   //**********************
216   // Codigos de retorno:
217   // -1: Final da Execucao
218   // 0: Termino da leitura de palavras no Grupo
219   // >0: Quantidade de palavras lidas.
220   int read_words(char st[100][max_size]) {
221     long long a, cn;
222
223     // Leitura da primeira palavra e validacao das condicoes de
          saida
224     scanf("%s", st[0]);
225     for (a = 0; a < strlen(st[0]); a++) st[0][a] = toupper(st[0][a
          ]);
226     if ((!strcmp(st[0], ":")) || (!strcmp(st[0], "EXIT")) || feof(
          stdin)) {
227       if ( (!strcmp(st[0], "EXIT")) || (feof(stdin)) ) return -1;
228       scanf("%s", st[0]);
```

```
229       if ( (!strcmp(st[0], "EXIT")) || (feof(stdin)) ) return -1;
230       return 0;
231     }
232
233     // Leitura das palavras seguintes
234     for (cn = 1; cn < 4; cn++) {
235       scanf("%s", st[cn]);
236       if ((!strcmp(st[cn], ":")) || (!strcmp(st[cn], "EXIT")) ||
             feof(stdin)) {
237         printf("\nERROR: Foram lidas %lld palavras, enquanto o
               esperado eram 4.\n", cn);
238         return -1;
239       }
240       for (a = 0; a < strlen(st[cn]); a++) st[cn][a] = toupper(st[
             cn][a]);
241     }
242
243     return 1;
244
245 }
246
247
248 int find_words (char st[100][max_size], long long *bi, char *
      vocab, long long words) {
249 //  long long a = 0, b = 0, cn = 0;
250     long long cn, b;
251
252     b = cn = 0;
253
254     // Procura pelas palavras no dicionario
255     for (cn = 0; cn < 4; cn++) {
256       for (b = 0; b < words; b++) if (!strcmp(&vocab[b * max_w],
             st[cn])) break;
257       bi[cn] = b;
258       if (bi[cn] == words) {
259         bi[cn] = 0;
260         return 0;
261       }
262 //      else
```

```
263 //       printf("\nWord: %s  Position in vocabulary: %lld\n", st[
        cn], bi[cn]);
264   }
265
266   return 1;
267 }
268
269
270 int find_nearest (long long words, float *vec, char *vocab,
        float *M, long long *bi, long long size, float *bestd, char
        bestw[N][max_size]) {
271   long long a, b, c, d;
272   float len, dist;
273
274   // Calculando o vetor resultante
275   for (a = 0; a < size; a++) vec[a] = M[a + bi[1] * size] - M[a
        + bi[0] * size] + M[a + bi[2] * size];
276   // Normalizando o vetor
277   len = 0;
278   for (a = 0; a < size; a++) len += vec[a] * vec[a];
279   len = sqrt(len);
280   for (a = 0; a < size; a++) vec[a] /= len;
281
282   for (a = 0; a < N; a++) bestd[a] = 0;
283   for (a = 0; a < N; a++) bestw[a][0] = 0;
284   for (c = 0; c < words; c++) {
285     if (c == bi[0]) continue;
286     if (c == bi[1]) continue;
287     if (c == bi[2]) continue;
288     a = 0;
289     for (b = 0; b < 3; b++) if (bi[b] == c) a = 1;
290     if (a == 1) continue;
291     dist = 0;
292     for (a = 0; a < size; a++) dist += vec[a] * M[a + c * size];
293     for (a = 0; a < N; a++) {
294       if (dist > bestd[a]) {
295         for (d = N - 1; d > a; d--) {
296           bestd[d] = bestd[d - 1];
297           strcpy(bestw[d], bestw[d - 1]);
```

```
298             }
299             bestd[a] = dist;
300             strcpy(bestw[a], &vocab[c * max_w]);
301             break;
302           }
303         }
304       }
305       return 1;
306     }
```