9.2.3 dstat

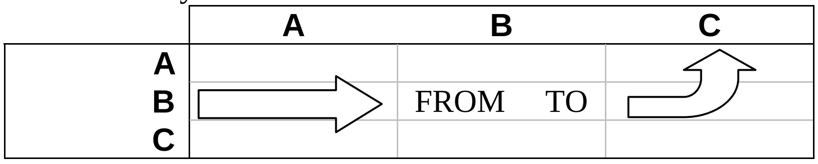
Application purpose

dstat will extract the uni-gram and bi-gram statistics for a data set, based on a {name}.pattern file. The .pattern file is in the format:

```
   <pattern num -- zero origin> <predecessor> <successor>
   <pattern num -- zero origin> <predecessor> <successor>
   .
   .
   .
   <EOF>
```

The predecessor and successor fields are numerical input identifiers. dstat is also supplied a “tags” file in order to associate the numerical identifiers with textual tags. When the data set statistics are output, these textual tags will be used to ease interpretation.

The transition arrays are formatted:



Usage

```
*** DataStat ver 0.22 -- Calculates dataset statistics
Usage: dstat {tagsFile} {patternFile} {outputBase}
Where: {tagsFile}    -- tags representing input lines
       {patternFile} -- file to specify output targets
       {outputBase}  -- base name of files to write output to
                        writes to {base}.unigram, {base}.bigram
                        (specify '-' for stdout)
For detailed info, type dstat --help
```

test.pattern

```
0 1 1
1 1 3
2 3 2
3 2 4
4 4 3
```

test.tags

```
A
B
C
D
```

The statistics are extracted by executing the command:

```
   dstat test.tags test.pattern test
```

```
- DataStat ver 0.22
 | Pattern file (inputs, targets) [test.pattern]
 +-> Output files [test].unigram, .bigram
Read 4 tags (4 outputs)
```

test.unigram

```
---- Unigram stats
 4 categories
 Unigram score: 50.00%
(category) (count) (percentage)
        A       1       16.67%
        B       1       16.67%
        C       3       50.00%
        D       1       16.67%
------------------------------
                6      100.00%
```

test.bigram

```
---- Bigram stats
 4 categories
 Bigram score: 83.33%
 - - - - - - - - - - - - - - - - - - - - - - - - -
                   A           B           C           D
        A          1           0           1           0
        B          0           0           0           1
        C          0           1           0           0
        D          0           0           2           0
------------------------------
   6 patterns total


                   A           B           C           D
        A      16.67%       0.00%      16.67%       0.00%
        B       0.00%       0.00%       0.00%      16.67%
        C       0.00%      16.67%       0.00%       0.00%
        D       0.00%       0.00%      33.33%       0.00%
------------------------------
   100% total




Most common transitions
------------------------------
        A          C
        B          D
        C          B
        D          C
------------------------------
```

Modules used

StdDefs, TokenLst, TokScan

Source code

Makefile

```
OBJFILES=tokenlst.o tokscan.o dstat.o
TARGET=dstat
VERSION=022
DISTFILES=tokenlst.c tokenlst.h tokscan.c tokscan.h dstat.c ${TARGET}
HELPFILE=${TARGET}.help

CC=gcc -g

CFLAGS=
LFALGS=

${TARGET}: ${OBJFILES}
   ${CC} -o ${TARGET} ${OBJFILES}

clean:
   rm -f *.o ${TARGET}

dist: ${DISTFILES}
   tar cvf ${TARGET}_v${VERSION}.tar ${DISTFILES}
   gzip ${TARGET}_v${VERSION}.tar

help: ${HELPFILE} ${TARGET}

${HELPFILE}:
   ${TARGET} --help

.c.o:
   ${CC} ${CFLAGS} ${LFLAGS} -c $*.c
```

dstat.c

```
/* Dataset statistics -- uses .tags and .pattern file
 *                        calculats unigram and bigram stats
 *
 * Author: Dylan Muir (dr.muir@student.qut.edu.au)
 *         QUT MLRC LPG August 1999
 * Date: 19th August, 1999
 * Modified: 12th October, 1999
 * Version: 0.22
 */

/* --- Includes --- */

#include <stdio.h>
#include <string.h>
#include <unistd.h>     /* for exit() */
#include <stdlib.h>
#include "tokenlst.h"
#include "tokscan.h"
#include "stddefs.h"


/* --- Defines --- */

#define VERSION   "0.22"
#define HELPFILE "dstat.help"

#define  APPNAME     0
#define  TAGSFILE 1
#define PATTERNFILE  2
#define  OUTPUTBASE  3
#define  NUMARGS     4

#define  MAXTAGLENGTH   10
#define  MAX_STRING  512


/* --- Helper functions' definitions --- */

void Usage(FILE *output, char *appName);
void  ProcessCommandLine(int  argc,  char  *argv[],  FILE  **tagsFile,  FILE  **  patternFile,  FILE
**unigramFile, FILE **bigramFile);
void Abort(int retnum);
void Help(char *appName);
void ReadTags(FILE *tagsFile, char **tagsArray[], int *numOutputs);
tokenList *ReadWholeFile(FILE *input);
void CloseFiles(FILE *tagsFile, FILE *patternFile, FILE *unigramFile, FILE *bigramFile);
void DoUnigramStats(FILE *patternFile, FILE *outputFile, int numOutputs, char *tagsArray[]);
void ReadPattern(FILE *patternFile, int *predecessor, int *successor, bool *inFile);
void  OutputUnigramStats(char  *tagsArray[],  long  outputCounts[],  FILE  *outputFile,  int  numOutputs,
int numPatterns, double unigramScore);
void DoBigramStats(FILE *patternFile, FILE *outputFile, int numOutputs, char *tagsArray[]);
void OutputBigramStats(char  *tagsArray[],  long  **bigramCounts,  int  *transitions,  FILE  *outputFile,
int numOutputs, int numPatterns, double bigramScore);
bool Allocate2DArray(char ***array, int xSize, int ySize, size_t elementSize);
int MaxLongArray(long array[], int numElements);


/* --- Main --- */

int main(int argc, char *argv[])
{
   FILE  *tagsFile, *patternFile, *unigramFile, *bigramFile;
   char  **tagsArray;
   int   numOutputs;

   ProcessCommandLine(argc, argv, &tagsFile, &patternFile, &unigramFile, &bigramFile);

   printf(" - DataStat ver %s\n", VERSION);
   printf(" | Pattern file (inputs, targets) [%s]\n", argv[PATTERNFILE]);
   printf(" +-> Output files [%s", argv[OUTPUTBASE]);

   unigramFile == stdout ? printf(" (stdout)]\n") : printf("].unigram, .bigram\n");

   ReadTags(tagsFile, &tagsArray, &numOutputs);
   printf("Read %d tags (%d outputs)\n", numOutputs, numOutputs);

   DoUnigramStats(patternFile, unigramFile, numOutputs, tagsArray);

   rewind(patternFile);
   DoBigramStats(patternFile, bigramFile, numOutputs, tagsArray);

   CloseFiles(tagsFile, patternFile, unigramFile, bigramFile);

   return 0;
}
```

```
/* --- Helper functions --- */

void Usage(FILE *output, char *appName)
{
   fprintf(output, "\n*** DataStat ver %s -- Calculates dataset statistics\n", VERSION);
   fprintf(output, "Usage: %s {tagsFile} {patternFile} {outputBase}\n", appName);
   fprintf(output, "Where: {tagsFile}    -- tags representing input lines\n");
   fprintf(output, "       {patternFile} -- file to specify output targets\n");
   fprintf(output, "       {outputBase}  -- base name of files to write output to\n");
   fprintf(output, "                        writes to {base}.unigram, {base}.bigram\n");
   fprintf(output, "                        (specify '-' for stdout)\n");
   fprintf(output, "For detailed info, type %s --help\n\n", appName);
}

void  ProcessCommandLine(int  argc,  char  *argv[],  FILE  **tagsFile,  FILE  **patternFile,  FILE
**unigramFile, FILE **bigramFile)
{
   char  buffer[MAX_STRING];

   if ((!(argc < 2)) && ((strcasecmp(argv[1], "--help") == 0))) {
      Help(argv[APPNAME]);
      exit(0);
   }

   if (argc < NUMARGS) {
      Usage(stderr, argv[APPNAME]);
      fprintf(stderr, "*** Invalid number of arguments.\n");
      Abort(1);
   }

   if (argc > NUMARGS)
      fprintf(stderr, "--- Warning: extra arguments ignored.\n");

   if ((*tagsFile = fopen(argv[TAGSFILE], "rt")) == NULL) {
      Usage(stderr, argv[APPNAME]);
      fprintf(stderr, "*** Could not open tags file [%s].\n", argv[TAGSFILE]);
      Abort(3);
   }


   if ((*patternFile = fopen(argv[PATTERNFILE], "rt")) == NULL) {
      Usage(stderr, argv[APPNAME]);
      fprintf(stderr, "*** Could not open pattern file [%s].\n", argv[PATTERNFILE]);
      Abort(5);
   }

   if (strcasecmp(argv[OUTPUTBASE], "-") == 0)
      *unigramFile = *bigramFile = stdout;
   else {
      strcpy(buffer, argv[OUTPUTBASE]);
      strcat(buffer, ".unigram");

      if ((*unigramFile = fopen(buffer, "wt")) == NULL) {
         Usage(stderr, argv[APPNAME]);
         fprintf(stderr, "*** Could not open unigram output file [%s].\n", buffer);
      }

      strcpy(buffer, argv[OUTPUTBASE]);
      strcat(buffer, ".bigram");

      if ((*bigramFile = fopen(buffer, "wt")) == NULL) {
         Usage(stderr, argv[APPNAME]);
         fprintf(stderr, "*** Could not open bigram output file [%s].\n", buffer);
      }
   }
}

void Abort(int retnum)
{
   fprintf(stderr, "Aborting...\n");
   exit(retnum);
}

void Help(char *appName)
{
   FILE *helpFile;

   if ((helpFile = fopen(HELPFILE, "wt")) == NULL) {
      fprintf(stderr, "*** Could not create help file [%s].\n", HELPFILE);
      Abort(2);
   }
   fprintf(helpFile, "----------------------------------------\n");
   fprintf(helpFile, "   Help file for DataStat ver %s\n", VERSION);
   fprintf(helpFile, "----------------------------------------\n\n");

   Usage(helpFile, appName);

   fprintf(helpFile, "DatasetStats will calculate and output the unigram and bigram statistics\n");
   fprintf(helpFile, "for for datasets represented by a .pattern file.\n\n");

   fprintf(helpFile, ".pattern file format:\n");
   fprintf(helpFile, "   (pattern num -- zero origin) (predecessor) (successor)\n");
   fprintf(helpFile, "   .\n");
   fprintf(helpFile, "   .\n");
   fprintf(helpFile, "   .\n");
   fprintf(helpFile, "   <EOF>\n\n");

   fprintf(helpFile, "DatasetStats will use a .tags file (tags separated by whitespace) to\n");
   fprintf(helpFile, "prettyprint the stats marked with the category names.\n\n");

   fprintf(helpFile, "DatasetStats is known to work on a 29-class problem successfully.\n");

   fprintf(stderr, "Created %s\n", HELPFILE);
}


void ReadTags(FILE *tagsFile, char **tagsArray[], int *numOutputs)
{
   tokenList   *allTags, *index;
   int      numTags;

   allTags = ReadWholeFile(tagsFile);

   if (allTags == NULL) {
      fprintf(stderr, "*** Error reading token file.\n");
      Abort(5);
   }

   numTags = 0;
   index = allTags;
   while (index != NULL) {
      index = index -> NEXT;
      numTags++;
   }

   *numOutputs = numTags;

   if (!Allocate2DArray(tagsArray, numTags, 1, MAXTAGLENGTH)) {
      fprintf(stderr, "*** Could not allocate tag array.\n");
      Abort(6);
   }

   index = allTags;
   numTags = 0;
   while (index != NULL) {
      strcpy((*tagsArray)[numTags], index -> token);
      index = index -> NEXT;
      numTags++;
   }
   DestroyTokenList(allTags);
}

tokenList *ReadWholeFile(FILE *input)
{
   tokenList   *list, *temp;
   bool     inFile;

   inFile = TRUE;
   list = ReadLineTokens(input, &inFile);   /* Get the first line */

   if (list == NULL)
      return list;

   while (inFile) {
      temp = ReadLineTokens(input, &inFile); /* Get the next line */
      list = ConcatenateTokenList(list, temp);
   }

   return list;
}


void CloseFiles(FILE *tagsFile, FILE *patternFile, FILE *unigramFile, FILE *bigramFile)
```

```
{
   fclose(tagsFile);
   fclose(patternFile);
   fclose(unigramFile);
   fclose(bigramFile);
}

void DoUnigramStats(FILE *patternFile, FILE *outputFile, int numOutputs, char *tagsArray[])
{
   long  *outputCounts, numPatterns;
   bool  inFile;
   int   predecessor, successor, outputIndex;

   if ((outputCounts = (long *) malloc(sizeof(long) * numOutputs)) == NULL) {
      fprintf(stderr, "*** Could not allocate unigram stats table.\n");
      Abort(3);
   }

   outputIndex = 0;
   while (outputIndex < numOutputs)
      outputCounts[outputIndex++] = 0;

   numPatterns = 0;
   inFile = TRUE;
   while (inFile) {
      ReadPattern(patternFile, &predecessor, &successor, &inFile);

      outputCounts[successor - 1]++;
      numPatterns++;
   }

   OutputUnigramStats(tagsArray,     outputCounts,     outputFile,     numOutputs,     numPatterns,
outputCounts[MaxLongArray(outputCounts, numOutputs)]);
}


void ReadPattern(FILE *patternFile, int *predecessor, int *successor, bool *inFile)
{
   int   patternNum;

   fscanf(patternFile, "%d %d %d", &patternNum, predecessor, successor);

   if (feof(patternFile))
      *inFile = FALSE;
}


void OutputUnigramStats(char *tagsArray[], long outputCounts[], FILE *outputFile, int numOutputs,
int numPatterns, double unigramScore){
   int   category;
   long  countSum;

   fprintf(outputFile, "---- Unigram stats\n");
   fprintf(outputFile, " %d categories\n", numOutputs);
   fprintf(outputFile, " Unigram score: %4.2f%%\n", unigramScore / (double) numPatterns * 100.0);
   fprintf(outputFile, "(category) (count) (percentage)\n");

   category = 0;
   while (category < numOutputs) {
      fprintf(outputFile,  "%9s    %6d    %10.2f%%\n",  tagsArray[category],  outputCounts[category],
(float) outputCounts[category] / (float) numPatterns * 100.0);
      category++;
   }

   fprintf(outputFile, "-----------------------------\n");
   fprintf(outputFile, "           %6d  %10.2f%%\n", numPatterns, 100.0);

}

void DoBigramStats(FILE *patternFile, FILE *outputFile, int numOutputs, char *tagsArray[])
{
   long  **bigramCounts, bigramScore;
   int   predecessor, successor, *transitions,
      numPatterns;
   bool  inFile;

   if (!Allocate2DArray((char ***) &bigramCounts, numOutputs, numOutputs, sizeof(long))) {
      fprintf(stderr, "*** Could not allocate bigram stats array.\n");
      Abort(4);
   }

   for (predecessor = 0; predecessor < numOutputs; predecessor++)
```

```
      for (successor = 0; successor < numOutputs; successor++)
         bigramCounts[predecessor][successor] = 0;

   numPatterns = 0;
   inFile = TRUE;
   while (inFile) {
      ReadPattern(patternFile, &predecessor, &successor, &inFile);

      bigramCounts[predecessor - 1][successor - 1]++;
      numPatterns++;
   }

   if ((transitions = (int *) malloc(sizeof(int) * numOutputs)) == NULL) {
      fprintf(stderr, "*** Could not allocate trainsitions array.\n");
      Abort(5);
   }

   bigramScore = 0;
   for (predecessor = 0; predecessor < numOutputs; predecessor++) {
      transitions[predecessor] = MaxLongArray(bigramCounts[predecessor], numOutputs);
      bigramScore += bigramCounts[predecessor][transitions[predecessor]];
   }

   OutputBigramStats(tagsArray,  bigramCounts,  transitions,  outputFile,  numOutputs,  numPatterns,
bigramScore);
}

void OutputBigramStats(char *tagsArray[], long **bigramCounts, int *transitions, FILE *outputFile,
int numOutputs, int numPatterns, double bigramScore)
{
   int   predecessor, successor;

   fprintf(outputFile, "---- Bigram stats\n");
   fprintf(outputFile, " %d categories\n", numOutputs);
   fprintf(outputFile, " Bigram score: %4.2f%%\n", bigramScore / (double) numPatterns * 100.0);

   fprintf(outputFile, " - - - - - - - - - - - - - - - - - - - - - - - - -\n");

   /* Counts */

   fprintf(outputFile, "            ");
   for (predecessor = 0; predecessor < numOutputs; predecessor++)
      fprintf(outputFile, "%9s   ", tagsArray[predecessor]);

   fprintf(outputFile, "\n");

   for (predecessor = 0; predecessor < numOutputs; predecessor++) {
      fprintf(outputFile, "%9s   ", tagsArray[predecessor]);

      for (successor = 0; successor < numOutputs; successor++)
         fprintf(outputFile, "%9d   ", bigramCounts[predecessor][successor]);

      fprintf(outputFile, "\n");
   }

   fprintf(outputFile, "------------------------------\n");
   fprintf(outputFile, "   %d patterns total\n\n\n", numPatterns);

   /* Percentages */

   fprintf(outputFile, "            ");
   for (predecessor = 0; predecessor < numOutputs; predecessor++)
      fprintf(outputFile, "%9s   ", tagsArray[predecessor]);

   fprintf(outputFile, "\n");

   for (predecessor = 0; predecessor < numOutputs; predecessor++) {
      fprintf(outputFile, "%9s   ", tagsArray[predecessor]);

      for (successor = 0; successor < numOutputs; successor++)
         fprintf(outputFile,  "%8.2f%%    ",  (float)  bigramCounts[predecessor][successor]  /  (float)
numPatterns * (float) 100);

      fprintf(outputFile, "\n");
   }

   fprintf(outputFile, "------------------------------\n");
   fprintf(outputFile, "   100%% total\n\n\n");

   fprintf(outputFile, "\n\n");
   fprintf(outputFile, "Most common transitions\n");
   fprintf(outputFile, "------------------------------\n");
   for (predecessor = 0; predecessor < numOutputs; predecessor++)
      fprintf(outputFile,        "%9s                        %9s\n",        tagsArray[predecessor],
tagsArray[transitions[predecessor]]);

   fprintf(outputFile, "------------------------------\n");
}

bool Allocate2DArray(char ***array, int xSize, int ySize, size_t elementSize)
{
   int   index;

   if ((*array = malloc(xSize * sizeof(void *))) == NULL)
      return FALSE;

   for (index = 0; index < xSize; index++)
      if (((*array)[index] = malloc(ySize * elementSize)) == NULL) {
         free (*array);
         return FALSE;
      }

   return TRUE;
}

int MaxLongArray(long array[], int numElements)
{
   long  max = 0.0;
   int   mindex = 0;

   while (numElements > 0) {
      if (array[numElements - 1] > max) {
         max = array[numElements - 1];
         mindex = numElements - 1;
      }

      numElements--;
   }

   return mindex;
}


/* --- END of dstat.c --- */
```