9.2.6 pattern

Application purpose

pattern generates .pattern files from tlearn .teach and .data files. .pattern files are used in several other applications, such as dstat and MakeFSA.

Usage

```
***pattern ver 0.01 -- creates pattern file from tlearn data
Usage: pattern {baseFileName}
       pattern {dataFile} {teachFile} {patternFile}
```

pattern can take as arguments either a base filename to both read the tlearn data from and to write the .pattern file to, or three separate filenames.

Modules used

StdDefs, TokenLst, TokScan,

Source code

Makefile

```
DISTFILES=pattern.c tokscan.c tokscan.h tokenlst.c tokenlst.h pattern
CC=gcc -g

pattern: pattern.o tokscan.o tokenlst.o
   ${CC} -o pattern pattern.o tokscan.o tokenlst.o
clean:
   rm -f *.o pattern
dist:
   tar cvf pattern_v001.tar $(DISTFILES)
   gzip pattern_v001.tar

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

pattern.c

```
/* -- Pattern.c -- make a .pattern file from tlearn data files
 *
 * Author: Dylan Muir (dr.muir@student.qut.edu.au)
 *         QUT MLRC LPG August 1999
 * Date: 9th August, 1999
 * Modified:
 * Version: 0.01
 */

/* --- Defines --- */

#define  VERSION        "0.01"
#define  APPNAME        0
#define  BASEFILENAME   1
#define  DATAFILE       1
#define  TEACHFILE      2
#define  PATTERNFILE    3
#define  NUMARGS        4
#define  MAX_STRING     512

#define  MIN(x, y)   ((x) > (y) ? (y) : (x))

/* --- Included headers --- */

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "tokenlst.h"
#include "tokscan.h"
#include "stddefs.h"


/* --- Enums --- */

typedef enum fileTypeEnum {distributed, localist, undefined} fileType;
```

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

void InterpretCommandLine(int argc, char *argv[]);
void Usage(FILE *output, char *argv[]);
void OpenFiles(int argc, char *argv[], FILE **dataFile, FILE **teachFile, FILE **patternFile);
void Abort(int errcode);
void CloseFiles(FILE *dataFile, FILE *teachFile, FILE *patternFile);
void ProcessFiles(FILE *dataFile, FILE *teachFile, FILE *patternFile);
fileType DetermineType(char *text);
void ReadFileInfo(FILE *file, fileType *fileStruct, int *numPatterns);
char *FileTypeString(fileType file);
void EatLine(FILE *file);


/* -- Main -- */

void main(int argc, char *argv[])
{
   FILE  *dataFile,
      *teachFile,
      *patternFile;

   InterpretCommandLine(argc, argv);
   OpenFiles(argc, argv, &dataFile, &teachFile, &patternFile);

   ProcessFiles(dataFile, teachFile, patternFile);

   CloseFiles(dataFile, teachFile, patternFile);
}


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

void InterpretCommandLine(int argc, char *argv[])
{
   switch (argc) {
      case 1:
      case 3:
         Usage(stderr, argv);
         fprintf(stderr, "*** Incorrect number of arguments.\n");
         Abort(1);
         break;

      default:
         break;
   }

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

void Usage(FILE *output, char *argv[])
{
   fprintf(output, "\n***pattern ver %s -- creates pattern file from tlearn data\n", VERSION);
   fprintf(output, "Usage: %s {baseFileName}\n", argv[APPNAME]);
   fprintf(output, "       %s {dataFile} {teachFile} {patternFile}\n", argv[APPNAME]);
}

void OpenFiles(int argc, char *argv[], FILE **dataFile, FILE **teachFile, FILE **patternFile)
{
   char  buffer[MAX_STRING],
      dataFName[MAX_STRING],
      teachFName[MAX_STRING],
      patternFName[MAX_STRING];

   switch(argc) {
      case 2:     /* Base name specified only */
         strcpy(dataFName, argv[BASEFILENAME]);
         strcpy(teachFName, argv[BASEFILENAME]);
         strcpy(patternFName, argv[BASEFILENAME]);

         strcat(dataFName, ".data");
         strcat(teachFName, ".teach");
         strcat(patternFName, ".pattern");

         break;

      case 4:
         strcpy(dataFName, argv[DATAFILE]);
         strcpy(teachFName, argv[TEACHFILE]);
```

```
         strcpy(patternFName, argv[PATTERNFILE]);

         break;

      default:
         fprintf(stderr, "*** Undefined error condition.\n");
         Abort(2);
   }

   printf(" - Pattern ver %s\n", VERSION);
   printf(" | Data file: [%s]\n", dataFName);
   printf(" | Teach file: [%s]\n", teachFName);
   printf(" +-> Pattern file: [%s]\n\n", patternFName);


   if ((*dataFile = fopen(dataFName, "rt")) == NULL) {
      fprintf(stderr, "*** Could not open [%s] for input.\n", dataFName);
      Abort(3);
   }

   if ((*teachFile = fopen(teachFName, "rt")) == NULL) {
      fprintf(stderr, "*** Could not open [%s] for input.\n", teachFName);
      fclose(*dataFile);
      Abort(3);
   }

   if ((*patternFile = fopen(patternFName, "wt")) == NULL) {
      fprintf(stderr, "*** Could not open [%s] for output.\n", patternFName);
      fclose(*dataFile);
      fclose(*teachFile);
      Abort(3);
   }
}

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

void CloseFiles(FILE *dataFile, FILE *teachFile, FILE *patternFile)
{
   if (dataFile != NULL)
      fclose(dataFile);

   if (teachFile != NULL)
      fclose(teachFile);

   if (patternFile != NULL)
      fclose(patternFile);
}


void ProcessFiles(FILE *dataFile, FILE *teachFile, FILE *patternFile)
{
   char     buffer[MAX_STRING];
   tokenList   *tokens;
   bool     inFile = TRUE;
   fileType dFileStruct, tFileStruct;
   int      dNumPatterns, tNumPatterns,
         patternIndex;

   ReadFileInfo(dataFile, &dFileStruct, &dNumPatterns);
   ReadFileInfo(teachFile, &tFileStruct, &tNumPatterns);

   printf(  "Data file: type[%s] patterns[%d]\n",
      FileTypeString(dFileStruct), dNumPatterns);
   printf(  "Teach file: type[%s] patterns[%d]\n",
      FileTypeString(tFileStruct), tNumPatterns);

   if (!(dFileStruct == localist && tFileStruct == localist)) {
      fprintf(stderr, "*** Error: file creation supported for localist files only.\n");
      Abort(5);
   }

   if (dNumPatterns > tNumPatterns)
      fprintf(stderr, "--- WARNING: extra patterns in .data will be ignored.\n");
   else if (dNumPatterns < tNumPatterns)
      fprintf(stderr, "--- WARNING: extra patterns in .teach will be ignored.\n");

   patternIndex =  0;
```

```
   while (  patternIndex < MIN(dNumPatterns, tNumPatterns) &&
      !feof(dataFile) && !feof(teachFile)) {

      fprintf(patternFile, "%d ", patternIndex);

      ReadLineText(dataFile, buffer, MAX_STRING, &inFile);
      fprintf(patternFile, "%s ", buffer);

      ReadLineText(teachFile, buffer, MAX_STRING, &inFile);
      fprintf(patternFile, "%s\n", buffer);

      patternIndex++;

      if ((patternIndex % 1000 == 0) && patternIndex > 10)
         printf("Processed %d patterns...\n", patternIndex);
   }

   if (patternIndex > 1000)
      printf("Processed %d patterns total.\n", patternIndex);
}

void ReadFileInfo(FILE *file, fileType *fileStruct, int *numPatterns)
{
   char  buffer[MAX_STRING];

   fscanf(file, "%s", buffer);
   EatLine(file);
   *fileStruct = DetermineType(buffer);

   fscanf(file, "%d", (int *) buffer);
   EatLine(file);
   *numPatterns = *(int *) buffer;
}


fileType DetermineType(char *text)
{
   if (strcmp(text, "distributed") == 0)
      return distributed;

   else if (strcmp(text, "localist") == 0)
      return localist;

   else
      return undefined;
}

char *FileTypeString(fileType file)
{
   switch(file) {
      case distributed:
         return "distributed";

      case localist:
         return "localist";

      case undefined:
         return "undefined";

      default:
         fprintf(stderr, "*** Uncategoriesed enum error.\n");
         Abort(4);
   }
}

void EatLine(FILE *file)
{
   char  read;

   read = 0x0;
   while (read != EOL && !feof(file))
      fread(&read, 1, 1, file);
}

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