Modulo 4 (apprendimento dei valori di SPAM) – terza bozza

In questo modulo dovete realizzare le funzioni necessarie ad esaminare i files contenuti in due directories che contengono le email da considerare HAM e quelle da considerare SPAM. Questo servirà a calcolare i valori di spam di ciascun token ed a creare il file del dizionario che viene usato dalle funzioni del modulo 3.

int calcolaPesoEmail(Token tokens[], int quanti, struct Dizionario * diz, double * peso);

Modificate la funzione implementata nel modulo 3 per tener conto del numero di occorrenze **occ_i** dello stesso token in **tok**. Il calcolo di **good_i** e **bad_i** dev'essere modificato come segue:

$$\mathbf{bad}_i = \prod [w_i * \mathbf{occ}_i] \qquad \mathbf{good}_i = \prod [(1 - w_i) * \mathbf{occ}_i]$$

Il resto del calcolo (sort, selezione, produttorie ecc.) rimane lo stesso.

int modificaValore(char * chiave, struct Dizionario * diz, double valore);

– modifica il valore associato alla chiave e se la chiave non c'è la inserisce

int prossimaChiave(char * chiave, struct Dizionario * diz, char ** next);

Cerca la chiave **next** che segue la chiave **chiave** (in un ordine da voi scelto).

Tornate la **prima** chiave se **chiave** è NULL. Tornate NO_MORE_ELEMENTS se non ci sono altre chiavi.

Tornate NOT_PRESENT se la **chiave** non è presente nel dizionario.

int scriviDizionario(char * filename, struct Dizionario * diz);

Scrive nel file **filename** le coppie di chiavi e valori contenute nel dizionario secondo il formato usato nel modulo 3, (in un ordine da voi scelto).

int leggiFiles(char * dirname, struct Dizionario * diz, int * numfiles);

Riempie il Dizionario **diz** con i tokens estratti nei files contenuti nella directory **dirname** contando il numero di occorrenze di ciascun token. Torna nella variabile **numfiles** il numero di files elaborati. **ATTENZIONE:** Per memorizzare il numero di occorrenze totali di ciascun token nei files usate i **valori** delle voci del dizionario.

int calcolaDizionario(struct Dizionario * HAM, int numh, struct Dizionario * SPAM, int nums, struct Dizionario * diz);

Riceve i due dizionari vettori di tokens HAM e SPAM che associano ad ogni token il suo numero di occorrenze come HAM e come SPAM. Riceve inoltre il numero di files esaminati **numh** (ham) e **nums** (spam).

Per ciascun token inserisce nel dizionario **diz** il peso calcolato secondo i passi che seguono:

- per ogni token **t_i**: sia **bad_i** il numero di occorrenze del token nello SPAM (e **good_i** nello HAM)
 - se (**bad_i + good_i**) è minore di 5 lo si ignora e si passa al prossimo
 - sia **b_i = bad_i / nums** la probabilità che il token sia spam (e **g_i = good_i / numh** che sia ham)
 - se le probabilità **b_i** e **g_i** sono maggiori di 1 le si pone a 1
 - il peso da assegnare è **w_i = b_i / (b_i + g_i)**
 - i valori di **w_i** minori di **0.01** vengono alzati a **0.01**
 - i valori di **w_i** maggiori di 0.99 vengono abbassati a 0.98 se **bad_i < 10**
 - i valori di **w_i** maggiori di 0.99 vengono abbassati a 0.99 se **bad_i >= 10**

int processDirs(char * hamdir, char * spamdir, char * filedizionario);

Riceve i due nomi delle directories che contengono le email HAM e SPAM, ed il nome del file dizionario da creare. Legge e tokenizza tutte le email contenute in **hamdir** e **spamdir**, calcola il dizionario e lo scrive nel file **filedizionario**.

Tests

Come al solito implementate i test contenuti in testModulo4.h