



Algoritmo di ottimizzazione “Correlation Feature Selection” in linguaggio assembly x86-32+SSE, x86-64+AVX e openMP

Progetto di Architetture e Programmazione dei Sistemi di Elaborazione

Gruppo 3: Antonino Vaccarella, Michele Purrone, Francesco Tolomeo

DIMES (Dipartimento di Ingegneria Informatica, Modellistica, Elettronica e Sistemistica)

Università della Calabria, Via Pietro Bucci, 87036, Arcavacata di Rende, CS, Italia

28 gennaio 2024

Sommario: Il presente elaborato ha l’obiettivo di mettere a punto un’implementazione dell’algoritmo Correlation Feature Selection in linguaggio C e di migliorarne le prestazioni utilizzando le tecniche di ottimizzazione basate sull’organizzazione dell’hardware. L’ambiente SW/HW di riferimento è costituito dal linguaggio di programmazione C (gcc), dal linguaggio assembly x86-32+SSE e dalla sua estensione x86-64+AVX (nasm) e dal sistema operativo Linux (Ubuntu).

1. Introduzione

Nel contesto contemporaneo dell’elaborazione dati, l’efficienza e l’efficacia nell’analisi di grandi dataset sono diventate imprescindibili. Il progetto in questione si pone in questo scenario con lo scopo di ottimizzare l’algoritmo **“Correlation Feature Selection”**.

L’algoritmo ha come obiettivo principale quello di selezionare un sottoinsieme di features a partire da un insieme più ampio, cercando di massimizzare la correlazione con la variabile di classe e al contempo minimizzare la correlazione tra le features stesse. L’idea chiave che sta alla base di CFS è che una feature andrebbe selezionata qualora sia altamente correlata con il target e contemporaneamente poco correlata con le altre features già selezionate, riducendo così la ridondanza. Tra i vantaggi dell’utilizzo di questo algoritmo si possono menzionare:

- **Riduzione della Dimensionalità:** selezionando un sottoinsieme ottimale di features viene ridotta la dimensionalità del dataset, portando quindi ad ottenere modelli più efficienti in termini di tempo computazionale e risorse;
- **Miglioramento della Generalizzazione:** la selezione accurata di features contribuisce a migliorare le prestazioni di generalizzazione del modello, riducendo inoltre il rischio di overfitting;
- **Interpretabilità:** avendo a che fare con una selezione di un numero più ridotto di features è possibile semplificare l’interpretazione del modello rendendo più facile l’identificazione di tutte quelle variabili che influenzano maggiormente le predizioni;
- **Riduzione del rumore:** CSF tende a rimuovere features fortemente collegate tra loro, riducendo il rumore nei dati e rendendo più stabile il modello.

ALGORITMO 2: Correlation Feature Selection

Input: un dataset DS definito sull'insieme di feature \mathcal{F} , un vettore c contenente le etichette, il numero k di feature da estrarre

Output: l'insieme $\mathcal{S} \subseteq \mathcal{F}$ delle k feature selezionate

```

1 begin
2   |    $\mathcal{S} = \emptyset;$ 
3   |   while  $|\mathcal{S}| < k$  do
4   |   |   calcolare, per ogni feature  $f_i$ , il punteggio  $merits_{\mathcal{S} \cup \{f_i\}}$  dell'insieme  $\mathcal{S} \cup \{f_i\}$ ;
5   |   |   sia  $f_i^*$  la feature che ha ottenuto il punteggio massimo;
6   |   |    $\mathcal{S} = \mathcal{S} \cup f_i^*$  ;
7   |   |   aggiungere all'insieme  $\mathcal{S}$  la feature  $f_i$  che ha ottenuto il punteggio massimo;

```

Pseudocodice

Per quanto riguarda l'algoritmo vero e proprio, si possono enumerare i seguenti passi principali:

1. **Inizializzazione:** l'algoritmo inizializza un insieme vuoto S senza selezionare alcuna caratteristica;
2. **Ciclo:** si imposta un ciclo che continua a iterare finché non è stato estratto il numero desiderato di caratteristiche (k);
3. **Corpo del ciclo:** ad ogni iterazione del ciclo, l'algoritmo esamina tutte le caratteristiche che non sono state ancora scelte e cerca di capire quale aggiungere all'insieme S . Per fare ciò, assegna un punteggio (*merito*) a ciascuna caratteristica, definito su quanto quest'ultima è correlata con le variabili di classe e quanto è unica rispetto alle caratteristiche che sono già state scelte. In particolare, ciò avviene con:
 - a) **Calcolo della Correlazione tra Features e Target:** si calcola la media delle correlazioni tra ciascuna feature e la variabile di classe;

$$r_{cf} = \frac{\mu_0 - \mu_1}{\sigma_f} \cdot \sqrt{\frac{n_0 \cdot n_1}{n^2}},$$

$$\sigma_f = \sqrt{\frac{1}{n-1} \cdot \sum_{i=1}^n (x_i - \mu)^2},$$

- b) **Calcolo della Correlazione tra le Features:** viene calcolata la media delle correlazioni tra tutte le coppie di features;

$$r_{f_x f_y} = \frac{\sum_{i=1}^n (x_i - \mu_x)(y_i - \mu_y)}{\sqrt{\sum_{i=1}^n (x_i - \mu_x)^2} \cdot \sqrt{\sum_{i=1}^n (y_i - \mu_y)^2}},$$

- c) **Calcolo del merito:** viene calcolato il merito dove $\overline{|r_{cf}|}$ è il valore medio del valore assoluto di tutte le correlazioni feature-classification e $\overline{|r_{ff}|}$ è il valore medio del

valore assoluto di tutte le correlazioni feature-feature;

$$merit_{S_k} = \frac{k \cdot \overline{|r_{cf}|}}{\sqrt{k + k \cdot (k - 1) \overline{|r_{ff}|}}},$$

4. **Scelta:** la caratteristica con il punteggio più alto viene selezionata e aggiunta all'insieme.

2. Scelte progettuali

2.1 Introduzione alle Scelte Progettuali

In questo capitolo, si analizzano le decisioni progettuali fondamentali adottate durante lo sviluppo del "Correlation Feature Selection".

Tali scelte rappresentano i pilastri del progetto, bilanciando l'efficienza computazionale con la leggibilità del codice e l'aderenza agli standard di programmazione. La scelta delle architetture a 32 bit e 64 bit, insieme alle implementazioni specifiche in Assembly x86-32 con SSE e x86-64 con AVX, è stata determinata da una valutazione accurata delle necessità computazionali e delle capacità di ciascuna piattaforma.

Si è prestata attenzione a ogni aspetto dell'algoritmo, dalla struttura del codice alle strategie di ottimizzazione. È stato cruciale scegliere formati di dati che garantissero un uso efficiente della memoria e un'elaborazione rapida, ottimizzando così le prestazioni complessive.

Di seguito, non solo si discute delle scelte tecniche ma anche delle sfide affrontate e delle strategie adottate per superarle, offrendo una comprensione completa del progetto ed evidenziando come ogni decisione sia stata essenziale nel raggiungimento degli obiettivi.

2.2 Principi Fondamentali

I principi fondamentali che hanno guidato le decisioni nel progetto combinano, come già accennato, efficienza, chiarezza e aderenza agli standard.

- **Efficienza Computazionale:** si è costantemente mirato all'ottimizzazione delle prestazioni. Lo sviluppo di un algoritmo funzionale ed efficiente sulle piattaforme target ha portato alla scelta di tecniche avanzate come SIMD, massimizzando le capacità delle architetture x86-32 e x86-64. Ci si è focalizzati su una rappresentazione efficiente dei dati, la minimizzazione dei cicli di clock e un utilizzo ottimizzato delle risorse della CPU;
- **Chiarezza del Codice:** pur enfatizzando l'efficienza, si è data grande importanza alla chiarezza del codice. Si è optato per un codice ben strutturato e commentato, fondamentale per la manutenibilità a lungo termine e la collaborazione tra i membri del team. Questo principio ha influenzato la scrittura di codice leggibile e organizzato, con nomi di variabili significativi e commenti dettagliati delle funzioni e dei loro scopi;
- **Aderenza agli Standard di Programmazione:** in ogni fase del progetto, si è mantenuta una stretta aderenza agli standard di programmazione. Ciò include l'osservanza delle best practices nella scrittura di codice Assembly e C.

In conclusione, questi principi rappresentano linee guida per il progetto, orientando ogni decisione verso la realizzazione di un algoritmo che non solo soddisfacesse i requisiti funzionali ma fosse anche ottimizzato, chiaro e conforme agli standard di qualità.

2.3 Scelte di Programmazione e Ottimizzazione

Nella realizzazione del progetto, si è deciso di adottare approcci di programmazione e ottimizzazione che hanno determinato in modo significativo la scalabilità e l'efficienza del sistema.

- **Assembly x86-32 con SSE per la Versione a 32 Bit:** l'uso dell'Assembly x86-32 con estensioni SSE è stata una scelta strategica per migliorare l'efficienza su architetture a 32 bit. L'impiego di SSE ha permesso operazioni parallele su vettori di dati, utilizzando i registri XMM a 128 bit. Ciò ha portato a una riduzione sostanziale dei tempi di calcolo, specialmente in operazioni voluminose e ripetitive. Questa scelta ha richiesto una programmazione precisa e mirata per massimizzare le potenzialità di elaborazione parallela;
- **Assembly x86-64 con AVX per la Versione a 64 Bit:** per la versione a 64 bit, l'adozione di AVX ha fornito un incremento significativo della capacità di elaborazione, grazie ai registri YMM a 256 bit. Ciò ha migliorato l'efficienza nella gestione di dataset complessi, consentendo un'elaborazione più rapida ed efficace. La scelta di AVX ha implicato una valutazione attenta degli impatti sulle prestazioni, in termini di uso della memoria e della CPU.

2.4 Rappresentazione dei Dati

La rappresentazione dei dati nel progetto è stata attentamente pianificata per massimizzare le potenzialità dell'algoritmo. Si è scelto di utilizzare array lineari in *column-major order* per le matrici di dati, basandosi su considerazioni di accessibilità e prestazioni. Questo formato facilita l'accesso sequenziale ai dati, ottimizzando l'uso della cache e riducendo i tempi di esecuzione.

- **Strutture di Dati e Formati:** per la versione a 32 bit, si è utilizzato il tipo di dato *float*, mentre per la versione a 64 bit il *double*, sfruttando così l'ottimizzazione nativa di questi tipi di dati con i set di istruzioni SSE e AVX.
- **Pratiche Ottimali e Alternative Considerate:** la scelta della rappresentazione in *column-major order* è stata influenzata dalla sua efficacia nelle operazioni matematiche e di calcolo intensivo. Anche se il *row-major order* era un'opzione valida, l'analisi ha mostrato che il *column-major order* era più compatibile con le sequenze di accesso ai dati previste dall'algoritmo. Per completezza, si riporta che sono state considerate strutture dati alternative, come array di array, ma si è concluso che gli array lineari offrivano una migliore performance in termini di accesso ai dati e gestione della memoria.

2.5 Parallelizzazione e Gestione delle Risorse

Nello sviluppo di "Correlation Feature Selection", si è posta particolare attenzione alla gestione efficiente delle risorse e all'implementazione di tecniche di parallelizzazione avanzate.

Tra le tecniche impiegate risultano:

- **Approccio di Parallelizzazione:** si è adottato un approccio basato sulle tecnologie SIMD sopracitate;
- **Loop Unrolling:** questa tecnica è stata utilizzata in ambito OpenMP per aumentare la velocità di esecuzione, riducendo i controlli di fine ciclo e unificando più istruzioni in un unico loop. Tale approccio ha minimizzato le penalità in caso di predizione errata da parte della BPU e, in presenza di istruzioni indipendenti, ha permesso di eseguirle in parallelo grazie alla presenza di più ALU nel processore.
- **Gestione e Ottimizzazione delle Risorse:** ci si è focalizzati sull'ottimizzazione dell'uso della memoria, garantendo un accesso più efficiente alla cache e riducendo i tempi di attesa del processore. L'introduzione di registri più ampi con AVX ha ulteriormente incrementato l'efficienza nelle operazioni su larga scala.

Le tecniche avanzate qui descritte richiedono un'attenta implementazione per massimizzare i benefici in termini di prestazioni, evitando eccessiva complessità o inefficienze. L'approccio strategicamente adottato ha dimostrato come un uso efficace delle capacità hardware possa migliorare significativamente l'efficienza di un algoritmo.

2.6 Test e Validazione

Per assicurare la correttezza e l'efficienza dell'algoritmo, si è adottato un rigoroso processo di test e validazione.

- **Test Unitari e Funzionali:** si sono utilizzati test unitari per la verifica di ogni componente e funzione dell'algoritmo, includendo il calcolo delle correlazioni, la gestione della memoria e le operazioni SIMD. I test funzionali hanno poi verificato il funzionamento complessivo del sistema, specialmente nell'elaborazione di set di dati di varie dimensioni e complessità.
- **Benchmarking e Profiling:** per valutare l'efficienza, si sono effettuati benchmarking estensivi, comparando le prestazioni dell'algoritmo nelle sue diverse implementazioni, misurando i tempi di esecuzione e analizzando il consumo di risorse. Il profiling ha aiutato a identificare e ottimizzare i colli di bottiglia, assicurando che l'algoritmo utilizzasse efficacemente le risorse hardware. Tra gli strumenti utilizzati figura “*gprof*”, un tool di analisi delle prestazioni per le applicazioni Unix. I comandi per utilizzare il tool sono: *gcc -pg -o cfs cfs.c* e *gprof cfs gmon.out > analysis.txt*.

```

1  Flat profile:
2
3  Each sample counts as 0.01 seconds.
4
5      % cumulative   self           self     total
6      time    seconds   seconds  calls  Ts/call  Ts/call  name
7
8      68.52    28.22   28.22
9      10.98    32.75   4.52
10     10.80    37.19   4.45
11     7.94     40.47   3.27
12     1.04     40.90   0.43
13     0.61     41.15   0.25
14     0.10     41.19   0.04
15     0.07     41.22   0.03
16     0.00     41.22   0.00      9    0.00    0.00  load_data
17     0.00     41.22   0.00      4    0.00    0.00  _mm_free
18     0.00     41.22   0.00      4    0.00    0.00  alloc_int_matrix
19     0.00     41.22   0.00      3    0.00    0.00  alloc_matrix
20     0.00     41.22   0.00      2    0.00    0.00  cfs
21
22      %           the percentage of the total running time of the
23      time        program used by this function.
24
25      cumulative a running sum of the number of seconds accounted
26      seconds     for by this function and those listed above it.
27
28      self        the number of seconds accounted for by this
29      seconds     function alone. This is the major sort for this
30      listing.
31
32      calls       the number of times this function was invoked, if
33      this function is profiled, else blank.
34
35      self        the average number of milliseconds spent in this
36      ms/call     function per call, if this function is profiled,
37      else blank.
38
39      total       the average number of milliseconds spent in this
40      ms/call     function and its descendants per call, if this
41      function is profiled, else blank.
42
43      name        the name of the function. This is the minor sort
44      for this listing. The index shows the location of
45      the function in the gprof listing. If the index is
46      in parenthesis it shows where it would appear in
47      the gprof listing if it were to be printed.
48 <0x0C>
49 Copyright (C) 2012-2022 Free Software Foundation, Inc.
50
51 Copying and distribution of this file, with or without modification,
52 are permitted in any medium without royalty provided the copyright
53 notice and this notice are preserved.
54 <0x0C>
55 Call graph (explanation follows)
56
57 granularity: each sample hit covers 2 byte(s) for 0.02% of 41.22 seconds
58
59      index % time   self   children   called   name
60      [1]   68.5  28.22   0.00
61      -----
62      [2]   11.0   4.52   0.00
63      -----
64      [3]   10.8   4.45   0.00
65      -----
66      [4]    7.9   3.27   0.00
67      -----
68      [5]    1.0    0.43   0.00
69      -----
70      [6]    0.6    0.25   0.00
71      -----
72      [7]    0.1    0.04   0.00
73      -----
74      [8]    0.1    0.03   0.00
75      -----
76      [9]    0.0
77
78      0.00   0.00   0.00   0.00   0.00   0.00   0.00
79      0.00   0.00   0.00   0.00   0.00   0.00   0.00
80      0.00   0.00   0.00   0.00   0.00   0.00   0.00
81      0.00   0.00   0.00   0.00   0.00   0.00   0.00
82      0.00   0.00   0.00   0.00   0.00   0.00   0.00
83      0.00   0.00   0.00   0.00   0.00   0.00   0.00
84      0.00   0.00   0.00   0.00   0.00   0.00   0.00
85      0.00   0.00   0.00   0.00   0.00   0.00   0.00
86      0.00   0.00   0.00   0.00   0.00   0.00   0.00
87      0.00   0.00   0.00   0.00   0.00   0.00   0.00
88      0.00   0.00   0.00   0.00   0.00   0.00   0.00
89      0.00   0.00   0.00   0.00   0.00   0.00   0.00
90      0.00   0.00   0.00   0.00   0.00   0.00   0.00

```

```

89                               3      load_data [9]
90 -----
91           0.00  0.00    4/4      load_data [9]
92 [10]     0.0   0.00    4      alloc_int_matrix [10]
93 -----
94           0.00  0.00    3/3      load_data [9]
95 [11]     0.0   0.00    3      alloc_matrix [11]
96 -----
97           0.00  0.00    2/2      _fini [41]
98 [12]     0.0   0.00    2      cfs [12]
99 -----
100          0.00  0.00    4/4      dealloc_matrix [18]
101 [33]     0.0   0.00    4      _mm_free [33]
102 -----
103
104 This table describes the call tree of the program, and was sorted by
105 the total amount of time spent in each function and its children.
106
107 Each entry in this table consists of several lines. The line with the
108 index number at the left hand margin lists the current function.
109 The lines above it list the functions that called this function,
110 and the lines below it list the functions this one called.
111 This line lists:
112     index A unique number given to each element of the table.
113     Index numbers are sorted numerically.
114     The index number is printed next to every function name so
115     it is easier to look up where the function is in the table.
116
117 % time This is the percentage of the 'total' time that was spent
118     in this function and its children. Note that due to
119     different viewpoints, functions excluded by options, etc,
120     these numbers will NOT add up to 100%.
121
122 self  This is the total amount of time spent in this function.
123
124 children  This is the total amount of time propagated into this
125     function by its children.
126
127 called This is the number of times the function was called.
128     If the function called itself recursively, the number
129     only includes non-recursive calls, and is followed by
130     a '+' and the number of recursive calls.
131
132 name  The name of the current function. The index number is
133     printed after it. If the function is a member of a
134     cycle, the cycle number is printed between the
135     function's name and the index number.
136
137
138 For the function's parents, the fields have the following meanings:
139
140 self  This is the amount of time that was propagated directly
141     from the function into this parent.
142
143 children  This is the amount of time that was propagated from
144     the function's children into this parent.

```

```

144      the function's children into this parent.
145
146      called This is the number of times this parent called the
147          function `/' the total number of times the function
148          was called. Recursive calls to the function are not
149          included in the number after the `/'.
150
151      name  This is the name of the parent. The parent's index
152          number is printed after it. If the parent is a
153          member of a cycle, the cycle number is printed between
154          the name and the index number.
155
156      If the parents of the function cannot be determined, the word
157          '<spontaneous>' is printed in the `name' field, and all the other
158          fields are blank.
159
160      For the function's children, the fields have the following meanings:
161
162          self  This is the amount of time that was propagated directly
163              from the child into the function.
164
165          children  This is the amount of time that was propagated from the
166              child's children to the function.
167
168          called This is the number of times the function called
169              this child `/' the total number of times the child
170              was called. Recursive calls by the child are not
171              listed in the number after the `/'.
172
173          name  This is the name of the child. The child's index
174              number is printed after it. If the child is a
175              member of a cycle, the cycle number is printed
176              between the name and the index number.
177
178      If there are any cycles (circles) in the call graph, there is an
179      entry for the cycle-as-a-whole. This entry shows who called the
180      cycle (as parents) and the members of the cycle (as children.)
181      The `+' recursive calls entry shows the number of function calls that
182      were internal to the cycle, and the calls entry for each member shows,
183      for that member, how many times it was called from other members of
184      the cycle.
185  <0x0c>
186  Copyright (C) 2012-2022 Free Software Foundation, Inc.
187
188  Copying and distribution of this file, with or without modification,
189  are permitted in any medium without royalty provided the copyright
190  notice and this notice are preserved.
191  <0x0c>
192  Index by function name
193
194      [33] _mm_free           [8] ciclo           [6] fine_ciclo_if01
195      [2] _mm_malloc          [3] ciclo_1          [9] load_data
196      [10] alloc_int_matrix   [5] ciclo_sommatoria SIMD  [4] main
197      [11] alloc_matrix        [7] correlation_ff
198      [12] cfs                [1] fine_ciclo

```

Analisi di gprof

- **Validazione Incrociata e Test di Robustezza:** si è implementata la validazione incrociata per garantire l'affidabilità dei risultati e condotto test di robustezza per verificare la resilienza dell'algoritmo in diverse condizioni operative, come variazioni nelle dimensioni dei set di dati.

3. Soluzioni intermedie

Lo sviluppo del progetto ha visto inevitabilmente il susseguirsi di diverse versioni. Tutto il percorso, dunque, per una maggiore chiarezza e per una questione coordinativa all'interno del gruppo, è stato raccolto in una repository GitHub, permettendo inoltre di tracciare un disegno molto accurato del progetto nel suo divenire.

Si espongono di seguito i principali passi nella realizzazione del software.

3.1 Alto livello

In primo luogo, ci si è focalizzati sull'implementazione dell'algoritmo unicamente per ciò che concerne l'uso di meccanismi di alto livello, ossia nel linguaggio C.

3.1.1 Versione solo C

In questa fase ci si è preoccupati dell'implementazione dell'algoritmo CFS, creando una versione in C senza richiamare alcuna procedura Assembly. Seguendo un approccio *bottom-up*, si è deciso di ordinare la matrice per colonna (*column-major order*) tramite una funzione per il calcolo della matrice trasposta (*transpose*) e successivamente si è seguita la struttura dello pseudocodice del Correlation Feature Selection (Capitolo 1).

In particolare, è stata implementata una funzione denominata “*correlation_cf*” che si occupa di misurare la correlazione tra una feature e la variabile di classe sfruttando il *point biserial correlation coefficient*. Successivamente è stata sviluppata una funzione “*correlation_ff*” che misura la correlazione tra una coppia di feature, ricorrendo alle formule matematiche definite per il *Pearson's correlation coefficient*.

La funzione *cfs*, infine, comprende un ciclo principale che viene eseguito tante volte quante sono le caratteristiche che si vogliono selezionare. Esso rappresenta le fasi “*Corpo del Ciclo*” e “*Scelta*” per come sono descritte nel Capitolo 1. In dettaglio, ad ogni iterazione del ciclo:

- la funzione valuta ogni caratteristica non ancora selezionata (*feature candidata*);
- calcola il merito per quella caratteristica;
- seleziona la caratteristica con il merito più alto.

Di seguito vengono riportati gli screenshots del codice.

```

209 //Deviazione standard
210 type stddev(type* x, int n) {
211     type mu = 0, sommatoria = 0, dev;
212     for(int i = 0; i<n; i++) {
213         mu += x[i];
214     }
215     mu = mu/n;
216
217     for(int i = 0; i<n; i++) {
218         sommatoria += (x[i] - mu)*(x[i] - mu);
219     }
220
221     dev = sqrt( (1.0/(n-1.0)) * sommatoria);
222     return dev;
223 }
```

Funzione per il calcolo della deviazione standard

```

225 //Funzione di correlazione feature-classe (point biserial correlation coefficient)
226 type correlation_cf(type* feature, type* labels, int n) {
227     type sum_0 = 0, sum_1 = 0, mu_0 = 0, mu_1 = 0, r_cf;
228     int n_0 = 0, n_1 = 0;
229
230     for(int i = 0; i < n; i++) {
231         if(labels[i] == 0) {
232             sum_0 += feature[i];
233             n_0++;
234         } else {
235             sum_1 += feature[i];
236             n_1++;
237         }
238     }
239
240     mu_0 = sum_0 / n_0;
241     mu_1 = sum_1 / n_1;
242
243     //DEVIAZIONE STANDARD
244     type dev = stddev(feature, n);
245
246     r_cf = ((mu_0 - mu_1) / dev) * sqrt((n_0 * n_1) / (type)(n * n));
247     return r_cf;
248 }
```

Funzione per il calcolo della correlazione class-feature, che richiama una funzione apposita per la deviazione standard

```

212 //Funzione di correlazione feature-classe (point biserial correlation coefficient)
213 type correlation_cf(type* feature, type* labels, int n) {
214     type sum_0 = 0, sum_1 = 0, mu_0 = 0, mu_1 = 0;
215     type sommatoria = 0, dev, r_cf;
216     int n_0 = 0, n_1 = 0;
217
218     // Calcolo delle medie per labels
219     for(int i = 0; i < n; i++) {
220         if(labels[i] == 0) {
221             sum_0 += feature[i];
222             n_0++;
223         } else {
224             sum_1 += feature[i];
225             n_1++;
226         }
227     }
228
229     mu_0 = sum_0 / n_0;
230     mu_1 = sum_1 / n_1;
231     type mu = (sum_0 + sum_1) / n;
232
233     // Pre-calcolo di valori costanti
234     type inv_n_minus_1 = 1.0 / (n - 1.0);
235     type sqrt_n0_n1_n_n = sqrt((n_0 * n_1) / (float)(n * n));
236
237     // Calcolo della deviazione standard in un ciclo
238     for(int i = 0; i < n; i++) {
239         sommatoria += (feature[i] - mu) * (feature[i] - mu);
240     }
241
242     dev = sqrt(inv_n_minus_1 * sommatoria);
243     r_cf = ((mu_0 - mu_1) / dev) * sqrt_n0_n1_n_n;
244     return r_cf;
245 }
```

Funzione per il calcolo della correlazione class-feature, che integra il calcolo della deviazione standard

```

247 //Funzione di correlazione feature-feature (Pearson's correlation coefficient)
248 type correlation_ff(type* x, type* y, int n) {
249     type sum_X = 0, sum_Y = 0, sum_XY = 0;
250     type squareSum_X = 0, squareSum_Y = 0;
251
252     for(int i = 0; i < n; i++) {
253         // Somma degli elementi della feature x
254         sum_X += x[i];
255         // Somma degli elementi della feature y
256         sum_Y += y[i];
257         // Somma dei prodotti
258         sum_XY += x[i] * y[i];
259         // Somma dei quadrati
260         squareSum_X += x[i] * x[i];
261         squareSum_Y += y[i] * y[i];
262     }
263
264     // Formula per il calcolo del coefficiente
265     type corr = (n * sum_XY - sum_X * sum_Y)
266     |   / sqrt((n * squareSum_X - sum_X * sum_X)
267     |   * (n * squareSum_Y - sum_Y * sum_Y));
268     return corr;
269 }
```

Funzione per il calcolo della correlazione feature-feature

```

271 void transpose(MATRIX A, MATRIX B, int rows, int cols) {
272     for(int i = 0; i < rows; ++i) {
273         for(int j = 0; j < cols; ++j) {
274             B[j * rows + i] = A[i * cols + j];
275         }
276     }
277 }
```

Funzione per il calcolo della matrice trasposta

```

279 //Funzione principale di Correlation Features Selection (CFS)
280 void cfs(params* input) {
281     VECTOR labels = input->labels; // etichette
282     int k = input->k;           // numero di features da estrarre
283     int rows = input->N;        // numero di righe del dataset
284     int cols = input->d;        // numero di colonne/feature del dataset
285
286     int selectedFeatures[k];
287     bool featureSelected[cols];
288     memset(featureSelected, 0, sizeof(featureSelected)); //Per azzerare
289
290     MATRIX dataset = alloc_matrix(rows, cols);
291     transpose(input->ds, dataset, rows, cols);
292
293     type r_cf = 0.0;
294     type r_cf_sum = 0.0;
295     type r_cf_den = 0.0;
296     type r_ff_sum = 0.0;
297     type r_ff_den = 0.0;
298
299     for(int s = 0; s < k; s++) {
300
301         type maxMerit = -1.0;
302         int maxFeature = -1;
303
304         for(int i = 0; i < cols; i++) {
305             if(!featureSelected[i]) {
306                 r_cf = correlation_cf(dataset + i * rows, labels, rows);
307
308                 for(int j = 0; j < s; j++) {
309                     r_cf_sum += fabs(correlation_cf(dataset + selectedFeatures[j] * rows, labels, rows));
310                     r_cf_den+=1.0;
311
312                     r_ff_sum += fabs(correlation_ff(dataset + selectedFeatures[j] * rows, dataset + i * rows, rows));
313                     r_ff_den++;
314
315                     for(int m = j + 1; m < s; m++) { // Inizia da j + 1 per evitare di ripetere le coppie
316                         r_ff_sum += fabs(correlation_ff(dataset + selectedFeatures[j] * rows, dataset + selectedFeatures[m] * rows, rows));
317                         r_ff_den++;
318                     }
319                 }
320
321                 r_cf_sum += fabs(r_cf);
322                 r_cf_den+=1.0;
323
324                 float K = (float) s + 1.0;
325
326                 // Calcolo del punteggio di merito per la feature i
327                 //merit_s_k caso 0 = {{k * [abs(r_cf)]} / {sqrt(k)}}
328                 //merit_s_k = {{k * [sommatoria i=0 to (k-1) [abs(r_cf)]/(k-1)]} / {sqrt(k + {k * (k-1) * [sommatoria i=0 to (k-1) [abs(r_ff)]/(k-1)]})}}
329                 float merit = (s == 0) ? (K * fabs(r_cf))/sqrt(K) :
330                 (K * (r_cf_sum / r_cf_den)) / sqrt(K + (K - 1.0) * (r_ff_sum / r_ff_den));
331
332                 r_cf_sum = 0.0;
333                 r_cf_den = 0.0;
334                 r_ff_sum = 0.0;
335                 r_ff_den = 0.0;
336                 r_cf = 0.0;
337
338                 if (merit > maxMerit) {
339                     maxMerit = merit;
340                     maxFeature = i;
341                 }
342             }
343         }
344         if(maxFeature != -1) {
345             featureSelected[maxFeature] = true;
346             selectedFeatures[s] = maxFeature;
347             printf("Questo è il merit della colonna: %d, e vale: %f, appartiene alla feature: %d\n", s, maxMerit, maxFeature);
348             input->sc = maxMerit;
349         }
350     }
351
352     // Copia i risultati selezionati in input->out
353     for (int i = 0; i < input->k; i++) {
354         input->out[i] = selectedFeatures[i];
355     }
356
357     dealloc_matrix(dataset);
358 }
359 }
```

Prima versione del codice interamente in linguaggio C

3.1.2 Versioni C migliorate

A seguito della prima versione, si è condotta un'attenta analisi del codice C alla ricerca di ulteriori miglioramenti.

Prima versione con chiamata alle procedure Assembly

Nel processo di ottimizzazione dell'algoritmo, si è presa la decisione strategica di sostituire alcune funzioni critiche, precedentemente implementate in C, con procedure equivalenti scritte in assembly x86-32 con supporto SSE e x86-64 con supporto AVX. La scelta è motivata dalla necessità di migliorare l'efficienza complessiva dell'algoritmo e di ridurre i tempi di esecuzione.

Le funzioni “*correlation_cf*”, “*correlation_ff*” e “*transpose*” sono state riscritte in assembly per sfruttare direttamente le istruzioni SIMD, permettendo di eseguire loop e operazioni matematiche su più dati contemporaneamente. L'approccio appena descritto è molto più efficace rispetto all'utilizzo di cicli iterativi in C, dove, a meno di adozione di specifiche tecniche di parallelizzazione, le operazioni vengono eseguite su un singolo elemento per volta.

I test di performance condotti dopo l'introduzione delle procedure assembly hanno dimostrato miglioramenti significativi nei tempi di esecuzione. Questi miglioramenti sono attribuibili all'efficienza con cui le istruzioni SIMD possono processare i dati e alla riduzione del sovraccarico dovuto alle chiamate di funzione e al contest switching tra il codice C e le operazioni di basso livello.

```

209 //Funzione principale di Correlation Features Selection (CFS)
210 void cfs(params* input) {
211
212     VECTOR labels = input->labels; // etichette
213     int k = input->k;           // numero di features da estrarre
214     int rows = input->N;        // numero di righe del dataset
215     int cols = input->d;        // numero di colonne/feature del dataset
216
217     int selectedFeatures[k];
218     bool featureSelected[cols];
219     memset(featureSelected, 0, sizeof(featureSelected)); //Per azzerare
220
221     MATRIX dataset = alloc_matrix(rows, cols);
222     transpose(input->ds, dataset, cols, rows);
223
224     type r_cf = 0.0;
225     type r_cf_sum = 0.0;
226     type r_cf_den = 0.0;
227     type r_ff = 0.0;
228     type r_ff_sum = 0.0;
229     type r_ff_den = 0.0;
230
231     for(int s = 0; s < k; s++) {
232
233         type maxMerit = -1.0;
234         int maxFeature = -1;
235
236         for(int i = 0; i < cols; i++) {
237             if(!featureSelected[i]) {
238                 correlation_cf(dataset + i * rows, labels, rows, &r_cf);
239                 r_cf_sum += fabs(r_cf);
240                 r_cf_den+=1.0;
241
242                 for(int j = 0; j < s; j++) {
243                     correlation_cf(dataset + selectedFeatures[j] * rows, labels, rows, &r_cf);
244                     r_cf_sum += fabs(r_cf);
245                     r_cf_den+=1.0;
246
247                     correlation_ff(dataset + selectedFeatures[j] * rows, dataset + i * rows, rows, &r_ff);
248                     r_ff_sum += fabs(r_ff);
249                     r_ff_den+=1.0;
250
251                     for(int m = j + 1; m < s; m++) { // Inizia da j + 1 per evitare di ripetere le coppie
252                         correlation_ff(dataset + selectedFeatures[j] * rows, dataset + selectedFeatures[m] * rows, rows, &r_ff);
253                         r_ff_sum += fabs(r_ff);
254                         r_ff_den+=1.0;
255                     }
256                 }
257
258                 type K = (type) s + 1.0;
259                 type merit = (s == 0) ? (K * fabs(r_cf))/sqrt(K) :
260                               (K * (r_cf_sum / r_cf_den)) / sqrt(K + (K * (K - 1.0) * (r_ff_sum / r_ff_den)));
261
262                 r_cf_sum = 0.0;
263                 r_cf_den = 0.0;
264                 r_ff_sum = 0.0;
265                 r_ff_den = 0.0;
266
267                 if (merit > maxMerit) {
268                     maxMerit = merit;
269                     maxFeature = i;
270                 }
271             }
272         }
273
274         if(maxFeature != -1) {
275             featureSelected[maxFeature] = true;
276             selectedFeatures[s] = maxFeature;
277             printf("Questo è il merit della colonna: %d, e vale: %f, appartiene alla feature: %d\n", s, maxMerit, maxFeature);
278             input->sc = maxMerit;
279         }
280     }
281
282     // Copia i risultati selezionati in input->out
283     for (int i = 0; i < input->k; i++) {
284         input->out[i] = selectedFeatures[i];
285     }
286
287     dealloc_matrix(dataset);
288 }

```

Prima versione con chiamate alle procedure Assembly

Seconda versione con il precalcolo dei coefficienti (CF e FF)

La seconda versione dell'algoritmo ha visto l'adozione di una strategia di precalcolo per le funzioni "*correlation_cf*" e "*correlation_ff*", permettendo di aumentare notevolmente l'efficienza complessiva del processo di selezione delle caratteristiche, e della configurazione della matrice trasposta nel *main*.

Invece di calcolare la correlazione tra le caratteristiche e le etichette (*correlation_cf*) e la correlazione tra le coppie di caratteristiche (*correlation_ff*) ogni volta che si rivelava necessario, si è preferito calcolare questi valori una sola volta all'inizio, per poi memorizzarli rispettivamente in un array e in una matrice. Tali strutture dati sono state poi utilizzate come riferimento per accedere rapidamente ai valori di correlazione precalcolati durante la selezione iterativa delle caratteristiche.

Il vantaggio di questo approccio è duplice: in primis, riduce il numero di calcoli ripetitivi, evitando così operazioni ridondanti e dispendiose in termini di tempo. In secondo luogo, accedere a un valore precalcolato in una struttura dati è molto più veloce che eseguire un'intera routine di calcolo, soprattutto quando si tratta di dataset di grandi dimensioni con un numero elevato di caratteristiche.

Questa scelta ha portato a una riduzione significativa dei tempi di esecuzione dell'algoritmo CFS. D'altro canto, la decisione ha anche richiesto una gestione attenta della memoria, dal momento che il precalcolo e la memorizzazione di un gran numero di correlazioni potrebbero richiedere una quantità sostanziale di spazio. Nonostante ciò, il trade-off tra l'uso della memoria e il miglioramento delle prestazioni si è dimostrato vantaggioso.

```

209 void cfs(params* input) {
210     VECTOR labels = input->labels; // etichette
211     int k = input->k;           // numero di features da estrarre
212     int rows = input->N;        // numero di righe del dataset
213     int cols = input->d;        // numero di colonne/feature del dataset
214     int selectedFeatures[k];
215     bool featureSelected[cols];
216     memset(featureSelected, 0, sizeof(featureSelected)); //Per azzerare
217     type r_cf = 0.0;
218     type r_cf_sum = 0.0;
219     type r_cf_den = 0.0;
220     type r_ff = 0.0;
221     type r_ff_sum = 0.0;
222     type r_ff_den = 0.0;
223     type cf_vector[cols];
224     for(int i = 0; i < cols; i++) {
225         correlation_cf(input->ds + i * rows, labels, rows, &r_cf);
226         cf_vector[i] = r_cf;
227     }
228     r_cf = 0.0;
229     type correlation_matrix[cols][cols];
230     for (int i = 0; i < cols; i++) {
231         for (int j = i + 1; j < cols; j++) { // j inizia da i + 1 per evitare calcoli ridondanti
232             correlation_ff(input->ds + i * rows, input->ds + j * rows, rows, &r_ff);
233             correlation_matrix[i][j] = r_ff;
234             correlation_matrix[j][i] = correlation_matrix[i][j]; // La correlazione è simmetrica
235         }
236     }
237     r_ff = 0.0;
238     for(int s = 0; s < k; s++) {
239         type maxMerit = -1.0;
240         int maxFeature = -1;
241         for(int i = 0; i < cols; i++) {
242             if(!featureSelected[i]) {
243                 r_cf = cf_vector[i];
244                 r_cf_sum += fabs(r_cf);
245                 r_cf_den+=1.0;
246                 for(int j = 0; j < s; j++) {
247                     r_cf = cf_vector[selectedFeatures[j]];
248                     r_cf_sum += fabs(r_cf);
249                     r_cf_den+=1.0;
250
251                     r_ff = correlation_matrix[i][selectedFeatures[j]];
252                     r_ff_sum += fabs(r_ff);
253                     r_ff_den+=1.0;
254                     for(int m = j + 1; m < s; m++) { // Inizia da j + 1 per evitare di ripetere le coppie
255                         r_ff = correlation_matrix[selectedFeatures[j]][selectedFeatures[m]];
256                         r_ff_sum += fabs(r_ff);
257                         r_ff_den+=1.0;
258                     }
259                 }
260                 type K = (type) s + 1.0;
261                 type merit = (s == 0) ? (K * fabs(r_cf))/sqrt(K) :
262                               (K * (r_cf_sum / r_cf_den)) / sqrt(K + (K * (K - 1.0) * (r_ff_sum / r_ff_den)));
263                 r_cf_sum = 0.0;
264                 r_cf_den = 0.0;
265                 r_ff_sum = 0.0;
266                 r_ff_den = 0.0;
267                 if (merit > maxMerit) {
268                     maxMerit = merit;
269                     maxFeature = i;
270                 }
271             }
272         }
273         featureSelected[maxFeature] = true;
274         selectedFeatures[s] = maxFeature;
275         input->sc = maxMerit;
276         input->out[s] = maxFeature;
277     }
278 }
```

Seconda versione con il precalcolo dei coefficienti (CF e FF)

```

374     input->ds = load_data(dsfilename, &input->N, &input->d);
375     MATRIX dataset_cmo = alloc_matrix(input->d, input->N);
376     transpose(input->ds, dataset_cmo, input->d, input->N);
377     dealloc_matrix(input->ds);
378     input->ds = dataset_cmo;

```

Configurazione del calcolo della matrice trasposta nel main

Terza versione con differenziazione del precalcolo

A seguito dei risultati ricavati dall'analisi teorica e da osservazioni empiriche ottenute da test approfonditi, un'ulteriore versione è stata implementata mediante l'introduzione di una logica condizionale che determina la strategia di precalcolo in base alle dimensioni del dataset e al numero di caratteristiche k da selezionare.

La scelta di calcolare anticipatamente le correlazioni tra le caratteristiche e le etichette e tra le coppie di caratteristiche, per poi memorizzarle in delle apposite strutture dati, è stata guidata dalla consapevolezza che tali calcoli, se eseguiti ripetutamente durante il processo di selezione, possono diventare una fonte significativa di inefficienza, specialmente con l'aumentare delle dimensioni del dataset e del numero di caratteristiche da estrarre.

Per i dataset più piccoli e per un k relativamente basso, la strategia di precalcolare solo “*correlation_cf*” ha mostrato un'eccellente efficienza, grazie alla riduzione del sovraccarico computazionale e di memoria. Tuttavia, per dataset più grandi e con un k più elevato, si è osservato che il beneficio di precalcolare anche “*correlation_ff*” superava di gran lunga il costo addizionale in termini di memoria, poiché eliminava la necessità di calcoli ridondanti in fase di esecuzione.

Di conseguenza, è stata implementata una soglia, calcolata empiricamente, che serve come punto di decisione per determinare quale strategia di precalcolo adottare. Quando k è al di sotto di questa soglia, l'algoritmo procede solo con il precalcolo di “*correlation_cf*”. Viceversa, quando k supera tale valore, si attiva il precalcolo completo della matrice per “*correlation_ff*”.

Questa soluzione ibrida offre il duplice vantaggio di mantenere una rapida esecuzione per scenari più piccoli e meno complessi, senza sacrificare l'efficienza nei casi in cui la complessità e le dimensioni del problema richiedano una gestione più robusta delle operazioni ripetitive.

```

215 //Funzione principale di Correlation Features Selection (CFS)
216 void cfs(params* input) {
217     VECTOR labels = input->labels; // etichette
218     int k = input->k;           // numero di features da estrarre
219     int rows = input->N;        // numero di righe del dataset
220     int cols = input->d;        // numero di colonne/feature del dataset
221     int selectedFeatures[k];
222     bool featureSelected[cols];
223     memset(featureSelected, 0, sizeof(featureSelected)); //Per azzerare
224     type r_cf = 0.0;
225     type r_cf_sum = 0.0;
226     type r_cf_den = 0.0;
227     type r_ff = 0.0;
228     type r_ff_sum = 0.0;
229     type r_ff_den = 0.0;
230     int soglia = 9.376 * log(0.063 * cols) - 4.584;
231     //Pilo il dataset è piccolo, pilo ci deve essere un valore piccolo a destra del k.
232     if(k<=soglia) {
233         printf("K formula: %d\n", soglia);
234         //Precalcolo di CF
235         type cf_vector[cols];
236         for(int i=0;i<cols;i++) {
237             correlation_cf(input->ds + i * rows, labels, rows, &r_cf);
238             cf_vector[i] = r_cf;
239         }
240         for(int s = 0; s < k; s++) {
241             type maxMerit = -1.0;
242             int maxFeature = -1;
243             type r_cf_sum_tmp = 0.0;
244             type r_ff_sum_tmp = 0.0;
245             r_cf_den += 1.0;
246             r_ff_den += (type) s;
247             type r_cf_sum_max = r_cf_sum;
248             type r_ff_sum_max = r_ff_sum;
249             for(int i = 0; i < cols; i++) {
250                 if(!featureSelected[i]) {
251                     r_cf_sum_tmp = r_cf_sum;
252                     r_ff_sum_tmp = r_ff_sum;
253                     r_cf = cf_vector[i];
254                     r_cf_sum_tmp += fabs(r_cf);
255                     for(int j = 0; j < s; j++) {
256                         correlation_ff(input->ds + i * rows, input->ds + selectedFeatures[j] * rows, rows, &r_ff);
257                         r_ff_sum_tmp += fabs(r_ff);
258                     }
259                     type K = (type) s + 1.0;
260                     type merit = (s == 0) ? (K * fabs(r_cf))/sqrt(K) :
261                         (K * (r_cf_sum_tmp / r_cf_den)) / sqrt(K + (K * (K - 1.0) * (r_ff_sum_tmp / r_ff_den)));
262                     if (merit > maxMerit) {
263                         maxMerit = merit;
264                         maxFeature = i;
265                         r_cf_sum_max = r_cf_sum_tmp;
266                         r_ff_sum_max = r_ff_sum_tmp;
267                     }
268                 }
269             }
270             r_cf_sum = r_cf_sum_max;
271             r_ff_sum = r_ff_sum_max;
272             featureSelected[maxFeature] = true;
273             selectedFeatures[s] = maxFeature;
274             input->sc = maxMerit;
275             input->out[s] = maxFeature;
276         }
277     }
278     else {
279         //Precalcolo di CF e FF sfruttando il for-i
280         type cf_vector[cols];
281         type correlation_matrix[cols][cols];
282         for (int i = 0; i < cols; i++) {
283             correlation_cf(input->ds + i * rows, labels, rows, &r_cf);
284             cf_vector[i] = r_cf;
285             for (int j = i + 1; j < cols; j++) { // j inizia da i + 1 per evitare calcoli ridondanti
286                 correlation_ff(input->ds + i * rows, input->ds + j * rows, rows, &r_ff);
287                 correlation_matrix[i][j] = r_ff;
288                 correlation_matrix[j][i] = correlation_matrix[i][j]; // La correlazione è simmetrica
289             }
}

```

```

289     }
290 }
291 r_ff = 0.0;
292 for(int s = 0; s < k; s++) {
293     type maxMerit = -1.0;
294     int maxFeature = -1;
295     type r_cf_sum_tmp = 0.0;
296     type r_ff_sum_tmp = 0.0;
297     r_cf_den += 1.0;
298     r_ff_den += (type) s;
299     type r_cf_sum_max = r_cf_sum;
300     type r_ff_sum_max = r_ff_sum;
301
302     for(int i = 0; i < cols; i++) {
303         if(!featureSelected[i]) {
304             r_cf_sum_tmp = r_cf_sum;
305             r_ff_sum_tmp = r_ff_sum;
306             r_cf = cf_vector[i];
307             r_cf_sum_tmp += fabs(r_cf);
308
309             for(int j = 0; j < s; j++) {
310                 r_ff = correlation_matrix[i][selectedFeatures[j]];
311                 r_ff_sum_tmp += fabs(r_ff);
312             }
313             type K = (type) s + 1.0;
314             type merit = (s == 0) ? (K * fabs(r_cf)) / sqrt(K) :
315             (K * (r_cf_sum_tmp / r_cf_den)) / sqrt(K + (K * (K - 1.0) * (r_ff_sum_tmp / r_ff_den)));
316             if (merit > maxMerit) {
317                 maxMerit = merit;
318                 maxFeature = i;
319                 r_cf_sum_max = r_cf_sum_tmp;
320                 r_ff_sum_max = r_ff_sum_tmp;
321             }
322         }
323     }
324     r_cf_sum = r_cf_sum_max;
325     r_ff_sum = r_ff_sum_max;
326     featureSelected[maxFeature] = true;
327     selectedFeatures[s] = maxFeature;
328     input->sc = maxMerit;
329     input->out[s] = maxFeature;
330 }
331 }
332 }
```

Terza versione con differenziazione del precalcolo

3.2 Basso livello

In questa sezione ci si è preoccupati dell'inserimento di procedure Assembly al fine di migliorare le prestazioni di quanto realizzato precedentemente in C. Relativamente alle ragioni prese in esame e in parte accennate, infatti, l'obiettivo è quello di aumentare la velocità di esecuzione servendosi del set istruzioni **SSE** e **AVX**. Affinché si possa essere in grado di lavorare con vettori di operandi anziché con un singolo operando. Si noti che il numero di operandi dipende chiaramente dal tipo di precisione adottata (i.e. float o double), ne conseguirà che il programma risulterà più veloce di un fattore approssimativamente pari a

$$\frac{\text{NormalExecutionTime}}{\text{OperandPerRegister}}$$

In altre parole, maggiore sarà il numero di operandi all'interno del vettore, maggiore sarà lo *speedup* ottenuto.

```

204 // PROCEDURE ASSEMBLY
205 extern void correlation_cf(type* feature, type* labels, int n, type* r_cf);
206 extern void correlation_ff(type* x, type* y, int n, type* r_ff);
207 extern void transpose(MATRIX A, MATRIX B, int rows, int cols);
```

4. Versioni finali

A seguito dell'analisi dei migliori risultati esposti nella sezione precedente, è stato individuato il risultato definitivo del progetto. Le specifiche di progetto richiedevano la realizzazione di quattro versioni complessive:

- una versione con operandi a 32 bit (*float*) con funzioni scritte in nasm utilizzando il set di istruzioni SSE;
- una versione con operandi a 32 bit che fa uso di openMP;
- una versione con operandi a 64 bit (*double*) con funzioni scritte in nasm utilizzando il set di istruzioni AVX;
- una versione con operandi a 64 bit che fa uso di openMP.

La struttura di tutte e quattro le versioni rimane invariata, basandosi sulla versione C migliorata precedentemente illustrata. Nel dettaglio, la transizione dalla versione a 32 bit a quella a 64 bit ha richiesto solo la modifica della dichiarazione del tipo a inizio file (modificando la direttiva `#define type` da *float* a *double*). Per quanto concerne la parte in Assembly, a causa del cambio di set di istruzioni, è stato necessario ridefinire le procedure, tenendo conto delle caratteristiche peculiari dei repertori SSE e AVX.

Per la versione OMP, si è deciso di:

1. aggiungere le direttive `#pragma OMP parallel for` nei punti più appropriati del codice C;
2. modificare la procedura Assembly per il calcolo della matrice trasposta, integrando il parallelismo tramite una nuova funzione C.

4.1 Linguaggio C

Nella versione finale di “Correlation Feature Selection”, si è optato per una strategia di ottimizzazione in grado di bilanciare efficacemente l’uso della memoria e il tempo di calcolo. In questa implementazione, la funzione “*correlation_cf*” continua a essere precalcolata per tutte le caratteristiche. L’approccio adottato si è dimostrato vantaggioso in termini di velocità, poiché il calcolo della correlazione class-feature viene effettuato almeno una volta nel ciclo principale di *cfs*, evitando le successive operazioni ridondanti.

Si è altresì rimosso il precalcolo della funzione “*correlation_ff*”. La decisione è stata presa a seguito di attente considerazioni circa la natura computazionalmente onerosa, soprattutto quando le dimensioni del dataset e il numero di caratteristiche da estrarre aumentano. Le correlazioni feature-feature vengono quindi calcolate e memorizzate nella matrice “*ff_matrix*” (inizializzata con valori fintizi), solo quando effettivamente necessario, ossia durante la valutazione di una nuova feature candidata da aggiungere all’insieme *S*. Nelle iterazioni successive alla prima si accede direttamente alla matrice contenente i valori ottenuti dal calcolo iniziale, risparmiando operazioni superflue.

Inoltre, l’uso di variabili temporanee come “*r_cf_sum_tmp*” e “*r_ff_sum_tmp*” permette di tenere traccia dei punteggi di merito durante il ciclo di selezione senza dover ricalcolare i valori già determinati in iterazioni precedenti. Le variabili non temporanee come “*r_cf_sum*” e “*r_ff_sum*” vengono aggiornate solo se la feature corrente si rivela essere una candidata valida per l’inclusione in *S*, minimizzando ulteriormente le operazioni di calcolo e risparmiando quindi un ciclo.

Le scelte progettuali intraprese consentono all'algoritmo di mantenere un'alta efficienza operativa. Ciò avviene in particolar modo in scenari dove il numero di caratteristiche da selezionare è molto elevato, potendo così gestire dataset di maggiori dimensioni senza un aumento esponenziale dei tempi di elaborazione.

```

190 //Funzione principale di Correlation Features Selection (CFS)
191 void cfs(params* input) {
192
193     //Dichiarazione variabili
194     int selectedFeatures[input->k]; //Array per tracciare le caratteristiche selezionate
195     bool featureSelected[input->d]; //Array booleano per marcare se una caratteristica è stata selezionata
196     memset(featureSelected, 0, sizeof(featureSelected)); //Inizializzazione di featureSelected a false
197
198     type cf_vector[input->d]; //Array per la correlazione class-feature
199
200     type **ff_matrix = (type **)malloc(input->d * sizeof(type *)); //Matrice per la correlazione feature-feature
201     for(int i = 0; i < input->d; i++) {
202         ff_matrix[i] = (type *)malloc(input->d * sizeof(type));
203     }
204
205     type r_cf = 0.0; //Correlazione class-feature
206     type r_cf_sum = 0.0; //Somma delle correlazioni class-feature
207     type r_cf_den = 0.0; //Denominatore per il calcolo del merito
208     type r_ff = 0.0; //Correlazione feature-feature
209     type r_ff_sum = 0.0; //Somma delle correlazioni feature-feature
210     type r_ff_den = 0.0; //Denominatore per il calcolo del merito
211
212     //Calcolo iniziale delle correlazioni
213     for (int i = 0; i < input->d; i++) {
214         correlation_cf(input->ds + i * input->N, input->labels, input->N, &r_cf);
215         cf_vector[i] = r_cf;
216         for (int j = 0; j < input->d; j++) {
217             ff_matrix[i][j] = -2.0;
218         }
219     }
220
221     //Ciclo di selezione delle caratteristiche
222     for(int s = 0; s < input->k; s++) {
223         type maxMerit = -1.0; //Variabile per tenere traccia del merito massimo
224         int maxFeature = -1; //Variabile per tenere traccia della miglior caratteristica
225         type r_cf_sum_tmp = 0.0;
226         type r_ff_sum_tmp = 0.0;
227         r_cf_den += 1.0; //Aggiornamento del denominatore class-feature
228         r_ff_den += (type) s; //Aggiornamento del denominatore feature-feature
229         type r_cf_sum_max = r_cf_sum;
230         type r_ff_sum_max = r_ff_sum;
231
232         //Ciclo interno per valutare ogni caratteristica non selezionata
233         for(int i = 0; i < input->d; i++) {
234             if(!featureSelected[i]) {
235                 r_cf_sum_tmp = r_cf_sum;
236                 r_ff_sum_tmp = r_ff_sum;
237                 r_cf = cf_vector[i];
238                 r_cf_sum_tmp += fabs(r_cf);
239
240                 //Calcolo delle correlazioni feature-feature per la caratteristica corrente
241                 for(int j = 0; j < s; j++) {
242                     if(ff_matrix[i][j] != -2.0) {
243                         r_ff_sum_tmp += ff_matrix[i][j];
244                     }
245                 else {
246                     correlation_ff(input->ds + i * input->N, input->ds + selectedFeatures[j] * input->N, input->N, &r_ff);
247                     type t = fabs(r_ff);
248                     r_ff_sum_tmp += t;
249                     ff_matrix[i][j] = t;
250                 }
251             }
252
253             //Calcolo del merito della caratteristica corrente
254             type K = (type) s + 1.0;
255             type merit = (s == 0) ? (K * fabs(r_cf))/sqrt(K) :
256             (K * (r_cf_sum_tmp / r_cf_den)) / sqrt(K + (K * (K - 1.0) * (r_ff_sum_tmp / r_ff_den)));
257
258             //Aggiornamento del massimo merito e della caratteristica corrispondente
259             if (merit > maxMerit) {
260                 maxMerit = merit;
261                 maxFeature = i;
262                 r_cf_sum_max = r_cf_sum_tmp;
263                 r_ff_sum_max = r_ff_sum_tmp;
264             }
265         }
266     }
267
268     //Aggiornamento delle variabili dopo la selezione della caratteristica
269     r_cf_sum = r_cf_sum_max;
270     r_ff_sum = r_ff_sum_max;
271     featureSelected[maxFeature] = true;
272     selectedFeatures[s] = maxFeature;
273     input->sc = maxMerit;
274     input->out[s] = maxFeature;
275 }
276
277 for (int i = 0; i < input->d; i++) { //Rilascio memoria
278     free(ff_matrix[i]);
279 }
280 free(ff_matrix);
281 }
```

4.2 x86-32+SSE

Come già discusso in precedenza, le funzioni implementate in Assembly sono:

- **void correlation_cf(type* feature, type* labels, int n, type* r_cf);**
- **void correlation_ff(type* x, type* y, int n, type* r_ff);**
- **void transpose(MATRIX A, MATRIX B, int rows, int cols);**

L'obiettivo nel corso del processo di sviluppo è stato la ricerca della maggiore ottimizzazione possibile, e, attraverso l'uso di istruzioni SIMD per operare su più elementi in parallelo si è evidenziato un notevole miglioramento per quanto concerne le prestazioni complessive.

correlation_cf

La funzione *correlation_cf* calcola il coefficiente di correlazione punto-biseriale (*point biserial correlation coefficient*) tra due vettori, uno rappresentante le caratteristiche (*feature*) e uno rappresentante le etichette dicotomiche (*labels*). La funzione accetta quattro parametri: gli indirizzi dei vettori *feature* e *labels*, la dimensione *n* del vettore (uguale per entrambe le strutture), e un puntatore *r_cf* dove verrà memorizzato il risultato del calcolo.

In primis, la procedura si occupa di inizializzare alcuni registri a zero, preparando al contempo (tramite le *C Calling Conventions*) i parametri su cui andrà a lavorare. Successivamente, esegue un ciclo su entrambi i vettori, sommando gli elementi delle caratteristiche in base ai valori corrispondenti nelle etichette. Le somme parziali vengono quindi utilizzate per calcolare le medie *mu_0* e *mu_1*.

Dopo aver calcolato le medie, la funzione procede calcolando la deviazione standard (*dev*) e altri termini di normalizzazione. Utilizzando istruzioni SIMD, viene eseguito un ciclo su parte del vettore *feature* per calcolare una sommatoria, riducendo il tempo di esecuzione.

La funzione gestisce poi eventuali elementi rimanenti nel vettore *feature* che non sono stati trattati con SIMD. La sommatoria complessiva viene quindi finalizzata e utilizzata per determinare il coefficiente di correlazione (*r_cf*) secondo la formula specificata dalle specifiche di progetto. Infine, il risultato viene salvato all'indirizzo di memoria fornito tramite il puntatore *r_cf*, e la funzione termina ripristinando i registri e lo stack.

```

46 ; ----- CORRELATION_CF -----
47 ;extern void correlation_cf(type* feature, type* labels, int n, type* r_cf);
48 section .text
49
50 global correlation_cf
51     ;feature      equ 8
52     ;labels       equ 12
53     ;n           equ 16
54     ;r_cf        equ 20
55
56 correlation_cf:
57     ; -----
58     ; Sequenza di ingresso nella funzione
59     ;
60     push    ebp
61     mov     ebp, esp
62     push    ebx
63     push    esi
64     push    edi
65
66     ;
67     ; FUNZIONE ASSEMBLY
68     ;
69     mov eax, [ebp+8]      ;feature addr
70     mov ebx, [ebp+12]     ;labels addr
71     mov ecx, [ebp+16]     ;n
72
73     xor esi, esi         ;index
74
75     xorps  xmm0, xmm0      ;sum 0
76     xorps  xmm1, xmm1      ;sum 1
77     xorps  xmm2, xmm2      ;mu 0
78     xorps  xmm3, xmm3      ;mu 1
79     xorps  xmm4, xmm4
80
81     xorps  xmm7, xmm7 ;vettore 0
82
83     xor edx, edx ;n_0
84     xor edi, edi ;n_1
85
86 ciclo_if01:
87     add esi, 1
88     cmp esi, ecx ;for...i < n...
89     jg fine_ciclo_if01
90
91     movss  xmm5, [eax+4*esi-4] ;feature[i]
92     movss  xmm6, [ebx+4*esi-4] ;labels[i]
93
94     ucomiss xmm6, xmm7 ;if(labels[i] == 0)
95     je increment_n0
96
97     jmp increment_n1
98
99 increment_n0:
100    addss xmm0, xmm5 ;sum_0 += feature[i]
101    inc edx ;n_0++
102    jmp ciclo_if01
103
104 increment_n1: ;else
105    addss xmm1, xmm5 ;sum_1 += feature[i]
106    inc edi ;n_1++
107    jmp ciclo_if01
108
109
110 fine_ciclo_if01:
111     movss  xmm2, xmm0
112     cvtsi2ss xmm4, edx      ;n_0 replicato 4 volte
113     divss  xmm2, xmm4 ;mu_0 = sum_0 / n_0
114
115     movss  xmm3, xmm1
116     cvtsi2ss xmm5, edi      ;n_1 replicato 4 volte
117     divss  xmm3, xmm5 ;mu_1 = sum_1 / n_1
118
119     ;type mu = (sum_0 + sum_1) / n
120     cvtsi2ss xmm7, ecx      ;n replicate 4 volte
121     addss xmm0, xmm1 ;sum_0 + sum_1

```

```

122     divss xmm0, xmm7 ;mu = (sum_0 + sum_1) / n
123
124     movaps xmm1, [uno]
125     subss xmm7, xmm1 ;(n - 1.0)
126     divss xmm1, xmm7 ;inv_n_minus_1 = 1.0 / (n - 1.0)
127
128     cvtsi2ss xmm7, ecx      ;n replicato 4 volte
129     mulss xmm7, xmm7 ;n * n
130     mulss xmm4, xmm5 ;n_0 * n_1
131     divss xmm4, xmm7 ;(n_0 * n_1) / (n * n)
132     sqrtss xmm4, xmm4 ;sqrt_n0_n1_n_n = sqrt((n_0 * n_1) / (type) (n * n))
133
134     shufps xmm0, xmm0, 0000000b
135     xor esi, esi      ;index
136     xorps xmm6, xmm6
137
138 ciclo_sommatoria SIMD:
139     add esi, 4
140     cmp esi, ecx ;for...i < n;...
141     jg fine_ciclo_sommatoria SIMD
142
143     movaps xmm5, [eax+4*esi-16]    ;feature[i]
144
145     subps xmm5, xmm0 ;(feature[i] - mu)
146
147     mulps xmm5, xmm5 ;(feature[i] - mu) * (feature[i] - mu)
148
149     addps xmm6, xmm5 ;sommatoria += (feature[i] - mu) * (feature[i] - mu)
150
151     jmp ciclo_sommatoria SIMD
152
153 fine_ciclo_sommatoria SIMD:
154     sub esi, 4
155
156 ciclo_sommatoria:
157     add esi, 1
158     cmp esi, ecx ;for...i < n;...
159     jg fine_ciclo_sommatoria
160
161     movss xmm5, [eax+4*esi-4] ;feature[i]
162     subss xmm5, xmm0 ;(feature[i] - mu)
163     mulss xmm5, xmm5 ;(feature[i] - mu) * (feature[i] - mu)
164
165     addss xmm6, xmm5 ;sommatoria += (feature[i] - mu) * (feature[i] - mu)
166
167     jmp ciclo_sommatoria
168
169 fine_ciclo_sommatoria:
170
171     haddps xmm6, xmm6
172     haddps xmm6, xmm6      ;prima cella sommatoria
173
174     mulss xmm1, xmm6 ;inv_n_minus_1 * sommatoria
175     sqrtss xmm1, xmm1 ;dev = sqrt(inv_n_minus_1 * sommatoria)
176
177     subss xmm2, xmm3 ;(mu_0 - mu_1)
178     divss xmm2, xmm1 ;((mu_0 - mu_1) / dev)
179     mulss xmm2, xmm4 ;r_cf = ((mu_0 - mu_1) / dev) * sqrt_n0_n1_n_n
180
181     mov edx, [ebp+20]
182     movss [edx], xmm2    ;ret corr value
183
184     ; -----
185     ; Sequenza di uscita dalla funzione
186     ; -----
187     pop edi
188     pop esi
189     pop ebx
190     mov esp, ebp
191     pop ebp
192     ret
193
194
195 ; ----- CORRELATION_CF -----
196

```

Procedura: *correlation_cf*

correlation_ff

La procedura *correlation_ff* è progettata per calcolare il coefficiente di correlazione di Pearson (*Pearson's correlation coefficient*) tra due vettori *x* e *y*, rappresentanti una coppia di caratteristiche (feature), ciascuno di lunghezza di *n*. Il risultato del calcolo viene memorizzato all'indirizzo di memoria specificato in *r_ff*.

Anche questa volta, la procedura inizia con l'inizializzazione dei registri e dei vettori necessari e la preparazione per l'esecuzione dei cicli di calcolo. Utilizzando istruzioni SIMD, viene eseguito un ciclo su entrambi i vettori (x e y), accumulando somme, prodotti e altri risultati intermedi in registri appositamente designati ($sumX$, $sumY$, $sumX*Y$, $sumX^2$, e $sumY^2$).

Se il numero totale di elementi non è un multiplo di quattro, viene eseguito un ciclo aggiuntivo per trattare gli elementi rimanenti. Dopo i cicli, le somme parziali vengono utilizzate per calcolare le quantità necessarie per la formula del coefficiente di correlazione. Applicata la formula, il risultato ottenuto viene quindi salvato all'indirizzo di memoria fornito attraverso il puntatore r_{ff} e la procedura termina ripristinando i registri e restituendo il controllo alla procedura chiamante.

```

197 ; ----- CORRELATION_FF -----
198 ;extern void correlation_ff(type* x, type* y, int n, type* r_ff);
199 section .text
200
201 global correlation_ff
202     ;x      equ 8
203     ;y      equ 12
204     ;n      equ 16
205     ;r_ff   equ 20
206
207 correlation_ff:
208     ; -----
209     ; Sequenza di ingresso nella funzione
210     ;
211     push    ebp
212     mov     ebp, esp
213     push    ebx
214     push    esi
215     push    edi
216
217     ;
218     ; FUNZIONE ASSEMBLY
219     ;
220     mov eax, [ebp+8]      ;x addr
221     mov ebx, [ebp+12]     ;y addr
222     mov ecx, [ebp+16]     ;n
223
224     xor esi, esi          ;index
225
226     xorps  xmm0, xmm0      ;sum X
227     xorps  xmm1, xmm1      ;sum Y
228     xorps  xmm2, xmm2      ;sum X*Y
229     xorps  xmm3, xmm3      ;square X
230     xorps  xmm4, xmm4      ;square Y
231
232 ciclo_SIMD:
233     add esi, 4
234     cmp esi, ecx
235     jg fine_ciclo_SIMD
236     movaps xmm5, [eax+4*esi-16]  ;x[i]
237     movaps xmm6, [ebx+4*esi-16]  ;y[i]
238     addps  xmm0, xmm5        ;sumX
239     addps  xmm1, xmm6        ;sumY
240     movaps xmm7, xmm5        ;copy x[i]
241     mulps  xmm7, xmm6        ;x[i]*y[i]
242     addps  xmm2, xmm7        ;sumX*Y
243     mulps  xmm5, xmm5        ;x[i]^2
244     mulps  xmm6, xmm6        ;y[i]^2
245     addps  xmm3, xmm5        ;sumX^2
246     addps  xmm4, xmm6        ;sumY^2
247
248     jmp ciclo_SIMD
249
250 fine_ciclo_SIMD:
251     sub esi, 4
252
253 ciclo_1:
254     add esi, 1
255     cmp esi, ecx
256     jg fine_ciclo
257     movss  xmm5, [eax+4*esi-4] ;x[i]
258     movss  xmm6, [ebx+4*esi-4] ;y[i]
259     addss  xmm0, xmm5        ;sumX
260     addss  xmm1, xmm6        ;sumY
261     movss  xmm7, xmm5        ;copy x[i]
262     mulss  xmm7, xmm6        ;x[i]*y[i]
263     addss  xmm2, xmm7        ;sumX*Y
264     mulss  xmm5, xmm5        ;x[i]^2
265     mulss  xmm6, xmm6        ;y[i]^2
266     addss  xmm3, xmm5        ;sumX^2
267     addss  xmm4, xmm6        ;sumY^2
268
269     jmp ciclo_1

```

```

270
271 fine_ciclo:
272     haddps xmm0, xmm0          ;prima cella sumX
273     haddps xmm0, xmm0          ;prima cella sumX
274     haddps xmm1, xmm1          ;prima cella sumY
275     haddps xmm1, xmm1          ;prima cella sumY
276     haddps xmm2, xmm2          ;prima cella sumX*Y
277     haddps xmm2, xmm2          ;prima cella sumX*Y
278     haddps xmm3, xmm3          ;prima cella sumX^2
279     haddps xmm3, xmm3          ;prima cella sumX^2
280     haddps xmm4, xmm4          ;prima cella sumY^2
281     haddps xmm4, xmm4          ;prima cella sumY^2
282
283     xorps  xmm5, xmm5
284     xorps  xmm6, xmm6
285     xorps  xmm7, xmm7
286
287 ;( N*xmm2 - xmm0 * xmm1 ) / sqrt[ ( N*xmm3 - xmm0*xmm0 ) * ( N*xmm4 - xmm1*xmm1 ) ]
288 ;   V1      V2      V3  V4      V5  V6
289 ;' _____'   ' _____'   ' _____'
290 ;   P1      P2      P3
291
292     cvtsi2ss xmm5, ecx      ;n replicato 4 volte
293     mulss  xmm2, xmm5          ;V1
294     movss  xmm6, xmm0
295     mulss  xmm6, xmm1          ;V2
296     mulss  xmm3, xmm5          ;V3
297     mulss  xmm0, xmm0          ;V4
298     mulss  xmm4, xmm5          ;V5
299     mulss  xmm1, xmm1          ;V6
300
301     subss  xmm2, xmm6          ;P1
302     subss  xmm3, xmm0          ;P2
303     subss  xmm4, xmm1          ;P3
304
305     mulss  xmm3, xmm4          ;contenuto sqrt
306     sqrtss xmm3, xmm3          ;sqrt
307
308     divss  xmm2, xmm3          ;corr value
309
310     mov edx, [ebp+20]
311     movss  [edx], xmm2          ;ret corr value
312
313 ;
314 ; Sequenza di uscita dalla funzione
315 ;
316     pop edi
317     pop esi
318     pop ebx
319     mov esp, ebp
320     pop ebp
321     ret
322 ; -----

```

CORRELATION_FF

Procedura: correlation_ff

transpose

La procedura *transpose* è progettata per calcolare la trasposta di una determinata matrice, sfruttando l'aritmetica degli indici per accedere agli elementi della matrice originale e copiarli nella posizione corretta della matrice trasposta. I parametri di ingresso includono l'indirizzo della matrice originale *A*, l'indirizzo della matrice trasposta *B*, il numero di colonne *c* e il numero di righe *r*.

La procedura inizia caricando gli indirizzi delle matrici di partenza e destinazione, e calcola il prodotto *c * r*, ovvero la lunghezza totale dell'array unidimensionale che rappresenta la matrice. Successivamente, inizia un ciclo principale (*ciclo*) che itera attraverso l'array unidimensionale rappresentante la matrice, calcolando gli indici *i* e *j* nella matrice originale in base all'indice corrente *n*.

Utilizzando questi indici, viene calcolato l'indice corrispondente nella matrice originale e il valore associato viene copiato nella matrice trasposta.

Il ciclo continua fino a quando l'indice *n* raggiunge o supera il prodotto *c * r*. Una volta completato il ciclo, la funzione termina.

```

324 ; ----- TRASPOSTA -----
325 ;extern void transpose(MATRIX A, MATRIX B, int ncols, int nrows);
326 section .text
327
328     A      equ 8
329     B      equ 12
330     c      equ 16
331     r      equ 20
332
333 global transpose
334
335 transpose:
336     ; -----
337     ; Sequenza di ingresso nella funzione
338     ;
339     push    ebp
340     mov     ebp, esp
341     push    ebx
342     push    esi
343     push    edi
344
345     ; -----
346     ; FUNZIONE ASSEMBLY
347     ;
348
349     mov esi, [ebp+8]
350     mov edi, [ebp+12]
351
352     mov eax, [ebp+16]
353     mov ecx, [ebp+20]
354     mul ecx
355     mov [cr], eax
356
357     xor eax, eax      ;indice n
358     mov [nx], eax
359
360 ciclo:
361     cmp eax, [cr]
362     jge fine
363
364     mov eax, [nx]
365     mov ebx, [ebp+r]      ;ebx=indiceRiga
366     cdq
367     div ebx            ;n/indiceRiga
368     mov [il], eax        ;i=n/indiceRiga
369     mov [jl], edx        ;j=n%indiceRiga
370
371     ;-----kt[n]=k[indiceCol*j]+i
372
373     xor eax, eax
374     xor ebx, ebx
375     xor ecx, ecx
376     mov eax, []
377     mov ebx, [ebp+c]      ;eax=indiceCol*j
378     mul ebx            ;eax=indiceCol*j
379     add eax, [il]
380     movss xmm0, [esi+eax*4]
381     mov ecx, [nx]
382     movss [edi+ecx*4], xmm0
383
384     ;-----incrementare n di 1
385
386     mov eax, [nx]
387     inc eax
388     mov [nx], eax
389     jmp ciclo
390
391 fine:
392     ; -----
393     ; Sequenza di uscita dalla funzione
394     ;
395     pop edi
396     pop esi
397     pop ebx
398     mov esp, ebp
399     pop ebp
400     ret
401 ; ----- TRASPOSTA -----
402

```

Procedura: trasposta

4.3 X86-64+AVX

La versione a 64 bit con il set di istruzioni AVX presenta molte analogie con la controparte a 32 bit con SSE. Si è reso necessario apportare alcune modifiche alle istruzioni per adattarle all'impiego di AVX ma, nonostante questi piccoli cambiamenti, la struttura generale degli algoritmi è rimasta sostanzialmente immutata.

Nello specifico, per evitare il *context switch* dovuto al passaggio da SSE ad AVX, si è optato per utilizzare operazioni riferite esclusivamente all'ultimo set di istruzioni. Anche nei casi in cui si sono impiegati i registri XMM di SSE, sono state adoperate le istruzioni AVX, facilmente identificabili dalla "V" iniziale. Avendo a disposizione vettori estesi a 256 bit e lavorando con operandi a 64 bit, il numero di elementi per vettore rimane invariato rispetto alla configurazione SSE esaminata in precedenza: quattro float nel caso precedente e quattro double nello scenario attuale.

È da sottolineare che, a differenza di SSE, laddove fosse possibile recuperare i parametri passati alla funzione attraverso il registro ESP (e pertanto sono presenti nello stack), in AVX i parametri vengono trasmessi alla funzione mediante appositi registri, il cui utilizzo e funzionamento sono specificati nelle *C Calling Conventions*.

Un altro aspetto degno di nota riguarda la riduzione dei vettori, non è infatti possibile utilizzare l'approccio precedentemente utilizzato. Si è quindi adottata la seguente strategia:

```
vhaddpd ymm8, ymm6, ymm6
vextractf128 xmm9, ymm8, 1
vaddsd xmm6, xmm9, xmm8
```

```
34 ; -----
35 ; Funzioni
36 ;
37
38 ; ----- CORRELATION_CF -----
39 ;extern void correlation_cf(type* feature, type* labels, int n, type* r_cf);
40 section .text
41
42 global correlation_cf
43     ;feature          rdi
44     ;labels           rsi
45     ;n                rdx
46     ;r_cf             rcx
47
48 correlation_cf:
49     ;
50     ; Sequenza di ingresso nella funzione
51     ;
52     push    rbp
53     mov     rbp, rsp
54     pushaq
55
56     ; -----
57     ; FUNZIONE ASSEMBLY
58     ;
59
60     vxorpd ymm0, ymm0, ymm0      ;sum_0
61     vxorpd ymm1, ymm1, ymm1      ;sum_1
62     vxorpd ymm2, ymm2, ymm2      ;mu_0
63     vxorpd ymm3, ymm3, ymm3      ;mu_1
64     vxorpd ymm4, ymm4, ymm4      ;vettore 0
65
66     xor r10, r10 ;n_0
67     xor r11, r11 ;n_1
68
69     xor    rax, rax           ;index
```

```

69          xor    rax, rax           ;index
70
71  ciclo_if01:
72          add    rax, 1
73          cmp    rax, rdx ;for(...i < n;...)
74          jg    fine_ciclo_if01
75
76          vmovsd xmm5, [rdi+8*rax-8]   ;feature[i]
77          vmovsd xmm6, [rsi+8*rax-8]   ;labels[i]
78
79          vcomisd xmm6, xmm4
80          je    increment_n0
81          jmp    increment_n1
82
83  increment_n0:
84          vaddsd xmm0, xmm5      ;sum_0 += feature[i]
85          add    r10, 1        ;n_0++
86          jmp    ciclo_if01
87
88  increment_n1: ;else
89          vaddsd xmm1, xmm5      ;sum_1 += feature[i]
90          add    r11, 1        ;n_1++
91          jmp    ciclo_if01
92
93
94  fine_ciclo_if01:
95
96          vcvttsi2sd xmm5, xmm5, r10      ;n_0 replicato
97          vcvttsi2sd xmm6, xmm6, r11      ;n_1 replicato
98          vcvttsi2sd xmm7, xmm7, rdx      ;n replicato
99
100         vmovsd xmm2, xmm0
101         vdivsd xmm2, xmm5      ;mu_0 = sum_0 / n_0
102
103         vmovsd xmm3, xmm1
104         vdivsd xmm3, xmm6      ;mu_1 = sum_1 / n_1
105
106         ;type mu = (sum_0 + sum_1) / n;
107         vaddsd xmm0, xmm1      ;sum_0 + sum_1
108         vdivsd xmm0, xmm7      ;mu = (sum_0 + sum_1) / n
109
110         vmovepd xmm1, [uno]
111         vmovepd xmm4, xmm7      ;n
112         vsubpd xmm4, xmm1      ;(n - 1.0)
113         vdivpd xmm1, xmm4      ;inv_n_minus_1 = 1.0 / (n - 1.0)
114
115         vmlusd xmm7, xmm7      ;n * n
116         vmlusd xmm5, xmm6      ;n_0 * n_1
117         vdivsd xmm5, xmm7      ;(n_0 * n_1) / (n * n)
118         vsqrtsd xmm5, xmm5      ;sqrt_n0_n1_n_n = sqrt((n_0 * n_1) / (type) (n * n))
119
120
121         vbroadcastsd ymm0, xmm0
122         vxorpd ymm4, ymm4, ymm4
123         vxorpd ymm6, ymm6, ymm6      ;sommatoria
124         xor    rax, rax           ;index
125
126  ciclo_sommatoriaSIMD:
127          add    rax, 4
128          cmp    rax, rdx ;for(...i < n;...)
129          jg    fine_ciclo_sommatoriaSIMD
130
131          vmovepd ymm4, [rdi+8*rax-32]   ;feature[i]
132          vsubpd ymm4, ymm0      ;(feature[i] - mu)
133          vmlulpd ymm4, ymm4      ;(feature[i] - mu) * (feature[i] - mu)
134          vaddpd ymm6, ymm4      ;sommatoria += (feature[i] - mu) * (feature[i] - mu);
135
136          jmp    ciclo_sommatoriaSIMD
137

```

```

138 fine_ciclo_sommatoriaSIMD:
139     sub    rax, 4
140
141     vxorpd ymm8, ymm8, ymm8
142     vxorpd ymm5, ymm9, ymm9
143     vhaddpd ymm8, ymm6, ymm6
144     vextractf128 xmm9, ymm8, 1
145     vaddsd xmm6, xmm6, xmm9, xmm8
146
147
148 ciclo_sommatoria:
149     add    rax, 1
150     cmp    rax, rdx ;for(...i < n;...)
151     jg    fine_ciclo_sommatoria
152
153     vmovsd xmm4, [rdi+8*rax-8] ;feature[i]
154     vsubsd xmm4, xmm0           ;(feature[i] - mu)
155     vmulsd xmm4, xmm4          ;(feature[i] - mu) * (feature[i] - mu)
156     vaddsd xmm6, xmm4          ;sommatoria += (feature[i] - mu) * (feature[i] - mu)
157
158     jmp ciclo_sommatoria
159
160 fine_ciclo_sommatoria:
161     vmulsd xmm1, xmm6          ;inv_n_minus_1 * sommatoria
162     vsqrtsd xmm1, xmm1         ;dev = sqrt(inv_n_minus_1 * sommatoria)
163
164     vsubsd xmm2, xmm3          ;(mu_0 - mu_1)
165     vdivsd xmm2, xmm1          ;((mu_0 - mu_1) / dev)
166     vmulsd xmm2, xmm5          ;r_cf = ((mu_0 - mu_1) / dev) * sqrt_n0_n1_n_n
167
168     vmovsd [rcx], xmm2        ;ret corr value
169
170     ; -----
171     ; Sequenza di uscita dalla funzione
172     ; -----
173     popaq
174     mov    rsp, rbp
175     pop    rbp
176     ret
177 ; ----- CORRELATION_CF -----
178

```

Procedura: correlation_cf

```

179 ; ----- CORRELATION_FF -----
180 ;extern void correlation_ff(type* x, type* y, int n, type* r_ff);
181 section .text
182
183 global correlation_ff
184
185 correlation_ff:
186     ; -----
187     ; Sequenza di ingresso nella funzione
188     ; -----
189     push    rbp
190     mov     rbp, rsp
191     pushaq
192
193     ; -----
194     ; FUNZIONE ASSEMBLY
195     ; -----
196
197 vxorpd ymm0, ymm0, ymm0 ; sum X
198 vxorpd ymm1, ymm1, ymm1 ; sum Y
199 vxorpd ymm2, ymm2, ymm2 ; sum X*Y
200 vxorpd ymm3, ymm3, ymm3 ; square X
201 vxorpd ymm4, ymm4, ymm4 ; square Y
202
203 xor    rax, rax    ; index
204
205 ciclo SIMD:
206
207 add    rax, 4
208 cmp    rax, rdx
209 jg     fine_ciclo SIMD
210 vmovapd ymm5, [rdi+8*rax-32] ; x[i]
211 vmovapd ymm6, [rsi+8*rax-32] ; y[i]
212 vaddpd ymm0, ymm0, ymm5      ; sumX
213 vaddpd ymm1, ymm1, ymm6      ; sumY
214 vmovapd ymm7, ymm5          ; copy x[i]
215 vmulpd ymm7, ymm7, ymm6      ; x[i]*y[i]
216 vaddpd ymm2, ymm2, ymm7      ; sumX*Y
217 vmulpd ymm5, ymm5, ymm5      ; x[i]^2
218 vmulpd ymm6, ymm6, ymm6      ; y[i]^2
219 vaddpd ymm3, ymm3, ymm5      ; sumX^2
220 vaddpd ymm4, ymm4, ymm6      ; sumY^2
221
222 jmp    ciclo SIMD
223
224 fine_ciclo SIMD:
225
226 sub    rax, 4
227
228 vxorpd ymm8, ymm8, ymm8
229 vxorpd ymm9, ymm9, ymm9
230 vhaddpd ymm8, ymm0, ymm0
231 vextractf128 xmm9, ymm8, 1
232 vaddsd xmm0, xmm9, xmm8
233
234 vhaddpd ymm8, ymm1, ymm1
235 vextractf128 xmm9, ymm8, 1
236 vaddsd xmm1, xmm9, xmm8
237
238 vhaddpd ymm8, ymm2, ymm2
239 vextractf128 xmm9, ymm8, 1
240 vaddsd xmm2, xmm9, xmm8
241
242 vhaddpd ymm8, ymm3, ymm3
243 vextractf128 xmm9, ymm8, 1
244 vaddsd xmm3, xmm9, xmm8

```

```

245
246     vhaddpd ymm8, ymm4, ymm4
247     vextractf128 xmm9, ymm8, 1
248     vaddsd xmm4, xmm9, xmm8
249
250
251 ciclo_1:
252     add    rax, 1
253     cmp    rax, rdx
254     jg     fine_ciclo
255     vmovsd xmm5, [rdi+8*rax-8] ; x[i]
256     vmovsd xmm6, [rsi+8*rax-8] ; y[i]
257     vaddsd xmm0, xmm0, xmm5 ; sumX
258     vaddsd xmm1, xmm1, xmm6 ; sumY
259     vmovsd xmm7, xmm5 ; copy x[i]
260     vmulsd xmm7, xmm7, xmm6 ; x[i]*y[i]
261     vaddsd xmm2, xmm2, xmm7 ; sumX*Y
262     vmulsd xmm5, xmm5, xmm5 ; x[i]^2
263     vmulsd xmm6, xmm6, xmm6 ; y[i]^2
264     vaddsd xmm3, xmm3, xmm5 ; sumY^2
265     vaddsd xmm4, xmm4, xmm6 ; sumY^2
266
267     jmp    ciclo_1
268
269 fine_ciclo:
270
271     vxorpd ymm5, ymm5, ymm5
272     vxorpd ymm6, ymm6, ymm6
273     vxorpd ymm7, ymm7, ymm7
274
275     ;( N*xmm2 - xmm0 * xmm1 ) / sqrt[ ( N*xmm3 - xmm0*xmm0 ) * ( N*xmm4 - xmm1*xmm1 ) ]
276     ;   V1      V2           V3      V4           V5      V6
277     ;'-----', '-----', '-----', '-----',
278     ;   P1          P2          P3
279
280     ; Calcolo della correlazione
281     vcvttsi2sd xmm5, xmm5, rdx ; n replicato 4 volte
282     vmulsd xmm2, xmm2, xmm5 ; V1
283     vmovsd xmm6, xmm0
284     vmulsd xmm6, xmm6, xmm1 ; V2
285     vmulsd xmm3, xmm3, xmm5 ; V3
286     vmulsd xmm0, xmm0, xmm0 ; V4
287     vmulsd xmm4, xmm4, xmm5 ; V5
288     vmulsd xmm1, xmm1, xmm1 ; V6
289
290     vsubsd xmm2, xmm2, xmm6 ; P1
291     vsubsd xmm3, xmm3, xmm0 ; P2
292     vsubsd xmm4, xmm4, xmm1 ; P3
293
294     vmulsd xmm3, xmm3, xmm4 ; contenuto sqrt
295     vsqrtsd xmm3, xmm3, xmm3 ; sqrt
296
297     vdivsd xmm2, xmm2, xmm3 ; corr value
298
299     vmovsd [rcx], xmm2 ; ret corr value
300
301     ; -----
302     ; Sequenza di uscita dalla funzione
303     ; -----
304     popaq
305     mov    rsp, rbp
306     pop    rbp
307     ret
308
309 ; ----- CORRELATION_FF -----
310

```

Procedura: correlation_ff

```

311 ; ----- TRASPOSTA -----
312
313 section .text
314     ;A      rdi
315     ;B      rsi
316     ;col    rdx
317     ;rig    rcx
318
319 global transpose
320
321 transpose:
322
323     ; -----
324     ; Sequenza di ingresso nella funzione
325     ;
326     push   rbp
327     mov    rbp, rsp
328     pushaq
329
330     ; -----
331     ; FUNZIONE ASSEMBLY
332     ;
333
334     xor   r12, r12      ; n = 0
335     mov   r13, rdx       ; r13 = col
336     mov   rax, r13       ; rax = col
337     mul   rcx          ; rax *= rig
338     mov   r9, rax        ; r9 = col*rig
339     ciclot: cmp   r12, r9      ; n < col*rig ?
340     jge   finet        ; se <= salta a finet
341     mov   rax, r12      ; rax = n
342     div   rcx          ; rax = n / rig, rdx = n % rig
343     mov   r10, rax       ; r10 = i
344     mov   r11, rdx       ; r11 = j
345     mov   rax, r13       ; rax = col
346     mul   r11          ; rax = col*j
347     add   rax, r10      ; rax += i
348     vmovsd xmm1, [rdi+rax*8]
349     vmovsd [rsi+r12*8], xmm1
350     inc   r12          ; n++
351     jmp   ciclot        ; salta ciclot
352
353 finet:
354     ; -----
355     ; Sequenza di uscita dalla funzione
356     ;
357     popaq
358     mov   rsp, rbp
359     pop   rbp
360     ret
361
362 ; ----- TRASPOSTA -----
363

```

Procedura: trasposta

4.4 Open MP

L'impiego di OpenMP rappresenta una strategia essenziale nell'implementazione del progetto, consentendo l'utilizzo di threads hardware durante l'esecuzione dei programmi. In questa fase, è molto importante raggiungere un equilibrato bilanciamento del carico di lavoro, evitare eventuali race conditions quando è necessario accedere a variabili condivise e garantire un corretto utilizzo del multithreading. Ciò implica la necessità di assicurare che il costo di creazione dei threads non superi il guadagno in termini di prestazioni, evitando così che i threads eseguano un numero limitato di operazioni.

A tale scopo, nel codice sono stati individuati i punti più adatti per l'introduzione del parallelismo. Questo è stato ottenuto sia identificando le zone che, per la natura stessa dell'algoritmo eseguito, si prestavano naturalmente al multi-threading, sia conducendo analisi sul codice per individuare le parti sottoposte a un maggiore stress computazionale.

```

204 void cfs(params* input) {
205
206     //Dichiarazione variabili
207     int selectedFeatures[input->k]; //Array per tracciare le caratteristiche selezionate
208     bool featureSelected[input->d]; //Array booleano per marcare se una caratteristica è stata selezionata
209     memset(featureSelected, 0, sizeof(featureSelected)); //Inizializzazione di featureSelected a false
210
211     type cf_vector[input->d]; //Array per la correlazione class-feature
212     type **ff_matrix = (type **)malloc(input->d * sizeof(type *)); //Matrice per la correlazione feature-feature
213     #pragma omp parallel for
214     for(int i = 0; i < input->d; i++) {
215         ff_matrix[i] = (type *)malloc(input->d * sizeof(type));
216     }
217
218     type r_cf = 0.0; //Correlazione class-feature
219     type r_cf_sum = 0.0; //Somma delle correlazioni class-feature
220     type r_cf_den = 0.0; //Denominatore per il calcolo del merito
221     type r_ff = 0.0; //Correlazione feature-feature
222     type r_ff_sum = 0.0; //Somma delle correlazioni feature-feature
223     type r_ff_den = 0.0; //Denominatore per il calcolo del merito
224
225     #pragma omp parallel for private(r_cf)
226     for (int i = 0; i < input->d; i++) {
227         correlation_cf(input->ds + i * input->N, input->labels, input->N, &r_cf);
228         cf_vector[i] = r_cf;
229
230         // Unrolling del loop interno
231         int j;
232         for (j = 0; j <= input->d - UNROLL_FACTOR; j += UNROLL_FACTOR) {
233             ff_matrix[i][j] = -2.0;
234             ff_matrix[i][j + 1] = -2.0;
235             ff_matrix[i][j + 2] = -2.0;
236             ff_matrix[i][j + 3] = -2.0;
237         }
238
239         // Gestione dei casi in cui input->d non è un multiplo di UNROLL_FACTOR
240         for (; j < input->d; j++) {
241             ff_matrix[i][j] = -2.0;
242         }
243     }
244
245     //Ciclo di selezione delle caratteristiche
246     for(int s = 0; s < input->k; s++) {
247         type maxMerit = -1.0; //Variabile per tenere traccia del merito massimo
248         int maxFeature = -1; //Variabile per tenere traccia della miglior caratteristica
249         type r_cf_sum_tmp = 0.0;
250         type r_ff_sum_tmp = 0.0;
251         r_cf_den += 1.0; //Aggiornamento del denominatore class-feature
252         r_ff_den += (type) s; //Aggiornamento del denominatore feature-feature
253         type r_cf_sum_max = r_cf_sum;
254         type r_ff_sum_max = r_ff_sum;
255
256         //Ciclo interno per valutare ogni caratteristica non selezionata
257         for(int i = 0; i < input->d; i++) {
258             if(!featureSelected[i]) {
259                 r_cf_sum_tmp = r_cf_sum;
260                 r_ff_sum_tmp = r_ff_sum;
261                 r_cf = cf_vector[i];
262                 r_cf_sum_tmp += fabs(r_cf);
263
264                 //Calcolo delle correlazioni feature-feature per la caratteristica corrente
265                 for(int j = 0; j < s; j++) {
266                     if(ff_matrix[i][j] != -2.0) {
267                         r_ff_sum_tmp += ff_matrix[i][j];
268                     }
269                     else {
270                         correlation_ff(input->ds + i * input->N, input->ds + selectedFeatures[j] * input->N, input->N, &r_ff);
271                         type t = fabs(r_ff);
272                         r_ff_sum_tmp += t;
273                         ff_matrix[i][j] = t;
274                     }
275                 }
276             }
277         }
278     }
279 }
```

```

277     //Calcolo del merito della caratteristica corrente
278     type K = (type) s + 1.0;
279     type merit = (s == 0) ? (K * fabs(r_cf))/sqrt(K) :
280     (K * (r_cf_sum_tmp / r_cf_den)) / sqrt(K + (K * (K - 1.0) * (r_ff_sum_tmp / r_ff_den)));
281
282     //Aggiornamento del massimo merito e della caratteristica corrispondente
283     if (merit > maxMerit) {
284         maxMerit = merit;
285         maxFeature = i;
286         r_cf_sum_max = r_cf_sum_tmp;
287         r_ff_sum_max = r_ff_sum_tmp;
288     }
289 }
290
291 //Aggiornamento delle variabili dopo la selezione della caratteristica
292 r_cf_sum = r_cf_sum_max;
293 r_ff_sum = r_ff_sum_max;
294 featureSelected[maxFeature] = true;
295 selectedFeatures[s] = maxFeature;
296 input->sc = maxMerit;
297 input->out[s] = maxFeature;
298 }
299
300 #pragma omp parallel for
301 for (int i = 0; i < input->d; i++) { //Rilascio memoria
302     free(ff_matrix[i]);
303 }
304 free(ff_matrix);
305
306 }
307 }
```

Versione con openMP

5. Confronto delle Prestazioni delle Diverse Versioni dell'Algoritmo CFS

L'ottimizzazione dell'algoritmo Correlation Feature Selection (CFS) ha visto diverse fasi di sviluppo, ognuna delle quali ha portato a specifici miglioramenti in termini di efficienza computazionale. Di seguito viene illustrato l'evolversi delle prestazioni attraverso le varie iterazioni dell'algoritmo.

Versione iniziale in linguaggio C: gli screenshot dei risultati di questa versione mostrano i tempi di esecuzione della pura implementazione in linguaggio C, senza ottimizzazioni di basso livello. Essi rappresentano il benchmark iniziale per le prestazioni.

```
└─$ ./run
./cfs32c -ds <DS> -labels <LABELS> -k <K> [-s] [-d]

Parameters:
    DS: il nome del file ds2 contenente il dataset
    LABELS: il nome del file ds2 contenente le etichette
    k: numero di features da estrarre

Options:
    -s: modo silenzioso, nessuna stampa, default 0 - false
    -d: stampa a video i risultati, default 0 - false
Dataset file name: 'test_5000_50_32.ds'
Labels file name: 'test_5000_50_32.labels'
Dataset row number: 5000
Dataset column number: 50
Number of features to extract: 5
Questo è il merit della colonna: 0, e vale: 0.025449, appartiene alla feature: 45
Questo è il merit della colonna: 1, e vale: 0.035685, appartiene alla feature: 25
Questo è il merit della colonna: 2, e vale: 0.042581, appartiene alla feature: 7
Questo è il merit della colonna: 3, e vale: 0.048312, appartiene alla feature: 33
Questo è il merit della colonna: 4, e vale: 0.053388, appartiene alla feature: 47
CFS time = 0.098 secs
sc: 0.053388, out: [45,25,7,33,47,]

Done.
```

Versione iniziale in linguaggio C

Versione Iniziale con Assembly: introducendo assembly x86-32 con SSE e x86-64 con AVX, sono stati ridotti i tempi di esecuzione rispetto alla versione puramente in C, sfruttando il controllo diretto sull'hardware e le istruzioni SIMD per processare dati in parallelo.

```

└─ $ ./run32
./cfs32c -ds <DS> -labels <LABELS> -k <K> [-s] [-d]

Parameters:
    DS: il nome del file ds2 contenente il dataset
    LABELS: il nome del file ds2 contenente le etichette
    k: numero di features da estrarre

Options:
    -s: modo silenzioso, nessuna stampa, default 0 - false
    -d: stampa a video i risultati, default 0 - false
Dataset file name: 'test_5000_50_32.ds'
Labels file name: 'test_5000_50_32.labels'
Dataset row number: 5000
Dataset column number: 50
Number of features to extract: 5
Questo è il merit della colonna: 0, e vale: 0.025449, appartiene alla feature: 45
Questo è il merit della colonna: 1, e vale: 0.035685, appartiene alla feature: 25
Questo è il merit della colonna: 2, e vale: 0.042581, appartiene alla feature: 7
Questo è il merit della colonna: 3, e vale: 0.048312, appartiene alla feature: 33
Questo è il merit della colonna: 4, e vale: 0.053388, appartiene alla feature: 47
CFS time = 0.038 secs
sc: 0.053388, out: [45,25,7,33,47,]

Done.

```

Versione iniziale con assembly

Versione con precalcolo di correlation_cf e correlation_ff: questa versione impiega un precalcolo simultaneo delle correlazioni CF e FF, migliorando la velocità per dataset più grandi e un numero elevato di caratteristiche k. Tuttavia, l'uso della memoria aumenta significativamente.

```

└─ $ ./run32
./cfs32c -ds <DS> -labels <LABELS> -k <K> [-s] [-d]

Parameters:
    DS: il nome del file ds2 contenente il dataset
    LABELS: il nome del file ds2 contenente le etichette
    k: numero di features da estrarre

Options:
    -s: modo silenzioso, nessuna stampa, default 0 - false
    -d: stampa a video i risultati, default 0 - false
Dataset file name: 'test_5000_50_32.ds'
Labels file name: 'test_5000_50_32.labels'
Dataset row number: 5000
Dataset column number: 50
Number of features to extract: 5
CFS time = 0.006 secs
sc: 0.053388, out: [45,25,7,33,47,]

Done.

```

Versione con precalcolo di CF e FF

Versione con precalcolo Condizionale e Soglia: implementando una logica condizionale basata su una soglia calcolata empiricamente, l'algoritmo esegue il precalcolo di CF sistematicamente e di FF solo quando k supera un certo valore. Questa scelta si riflette in un bilanciamento ottimale tra l'uso della memoria e il tempo di esecuzione.

```
└─ $ ./run32
./cfs32c -ds <DS> -labels <LABELS> -k <K> [-s] [-d]

Parameters:
  DS: il nome del file ds2 contenente il dataset
  LABELS: il nome del file ds2 contenente le etichette
  k: numero di features da estrarre

Options:
  -s: modo silenzioso, nessuna stampa, default 0 - false
  -d: stampa a video i risultati, default 0 - false
Dataset file name: 'test_75000_1350_32.ds'
Labels file name: 'test_75000_1350_32.labels'
Dataset row number: 75000
Dataset column number: 1350
Number of features to extract: 135
CFS time = 39.800 secs
sc: 0.075206, out: [1265,405,844,199,1337,45,13,328,629,1320,918,855,358,1345,265,1313,32,1385,216,763,184,489,637,991,1076,784,649,915,587,78,686,1056,1286,1244,442,494,94,586,845,810,2,252
,738,1207,525,691,80,723,1346,1316,432,1283,522,1088,371,484,1163,438,838,528,647,1022,820,74,974,718,1074,221,106,893,697,55,866,795,12,520,748,457,121,248,125,1026,1084,1053,1012,68,727,66
,1298,1043,21,1024,898,673,480,1189,764,1256,1157,1087,108,1219,1289,114,635,617,660,875,102,1259,623,989,1284,1323,160,277,232,68,1093,1326,1092,1124,346,737,951,159,1212,1234,183,940,437,
568,179,240,462,]

Done.
```

Versione con ff precalcolato

```
└─ $ ./run32
./cfs32c -ds <DS> -labels <LABELS> -k <K> [-s] [-d]

Parameters:
  DS: il nome del file ds2 contenente il dataset
  LABELS: il nome del file ds2 contenente le etichette
  k: numero di features da estrarre

Options:
  -s: modo silenzioso, nessuna stampa, default 0 - false
  -d: stampa a video i risultati, default 0 - false
Dataset file name: 'test_75000_1350_32.ds'
Labels file name: 'test_75000_1350_32.labels'
Dataset row number: 75000
Dataset column number: 1350
Number of features to extract: 135
CFS time = 443.602 secs
sc: 0.075206, out: [1265,405,844,199,1337,45,13,328,629,1320,918,855,358,1345,265,1313,32,1385,216,763,184,489,637,991,1076,784,649,915,587,78,686,1056,1286,1244,442,494,94,586,845,810,2,252
,738,1207,525,691,80,723,1346,1316,432,1283,522,1088,371,484,1163,438,838,528,647,1022,820,74,974,718,1074,221,106,893,697,55,866,795,12,520,748,457,121,248,125,1026,1084,1053,1012,68,727,66
,1298,1043,21,1024,898,673,480,1189,764,1256,1157,1087,108,1219,1289,114,635,617,660,875,102,1259,623,989,1284,1323,160,277,232,68,1093,1326,1092,1124,346,737,951,159,1212,1234,183,940,437,
568,179,240,462,]

Done.
```

Versione senza ff precalcolato

Versione finale: nella versione finale, è stato rimosso il precalcolo di FF mantenendo quello di CF, con un caching intelligente delle correlazioni già calcolate. Questo approccio riduce ulteriormente i tempi di esecuzione mantenendo un uso efficiente della memoria. La versione finale si distingue come la più performante tra quelle testate.

```
└─ $ ./run32
./cfs32c -ds <DS> -labels <LABELS> -k <K> [-s] [-d] Progetto
                                         ↓
                                         Parameters:
                                         DS: il nome del file ds2 contenente il dataset
                                         LABELS: il nome del file ds2 contenente le etichette
                                         k: numero di features da estrarre
                                         ↓
                                         Options:
                                         -s: modo silenzioso, nessuna stampa, default 0 - false
                                         -d: stampa a video i risultati, default 0 - false
Dataset file name: 'test_5000_50_32.ds'
Labels file name: 'test_5000_50_32.labels'
Dataset row number: 5000
Dataset column number: 50
Number of features to extract: 5
CFS time = 0.002 secs
sc: 0.053388, out: [45,25,7,33,47,]
Done.
```

```
./run64
./cf564c -ds <DS> -labels <LABELS> -K <K> [-s] [-d]

Parameters:
  DS: il nome del file ds2 contenente il dataset
  LABELS: il nome del file ds2 contenente le etichette
  K: numero di features da estrarre

Options:
  -s: modo silenzioso, nessuna stampa, default 0 - false
  -d: stampa a video i risultati, default 0 - false
Trasposta time = 0.005 secs
Ho eseguito la trasposta!
Dataset file name: 'test_5000_50_64.ds'
Labels file name: 'test_5000_50_64.labels'
Dataset row number: 5000
Dataset column number: 50
Number of features to extract: 5
CFS time = 0.003 secs
sc: 0.053390, out: [45,25,7,33,47,]

Done.
```

```
./run32
./cf532c -ds <DS> -labels <LABELS> -K <K> [-s] [-d]

Parameters:
  DS: il nome del file ds2 contenente il dataset
  LABELS: il nome del file ds2 contenente le etichette
  K: numero di features da estrarre

Options:
  -s: modo silenzioso, nessuna stampa, default 0 - false
  -d: stampa a video i risultati, default 0 - false
Dataset file name: 'test_75000_1350_32.ds'
Labels file name: 'test_75000_1350_32.labels'
Dataset row number: 75000
Dataset column number: 1350
Number of features to extract: 135
CFS time = 7.147 secs
sc: 0.075206, out: [1265,405,844,199,1337,45,13,328,629,1320,918,855,358,1345,265,1313,32,1305,216,763,184,489,637,991,1076,784,649,915,587,78,686,1056,1286,1244,442,494,94,586,845,810,2,252,738,1207,525,691,80,723,1346,1316,432,1283,522,1088,371,484,1163,438,838,528,647,1022,820,74,974,718,1074,221,106,893,697,55,866,795,12,520,748,457,121,248,125,1026,1084,1053,1012,68,727,66,1298,1843,21,1024,898,673,400,1189,764,1256,1157,1087,108,1219,1289,114,635,617,666,875,102,1259,623,989,1204,1323,160,277,232,88,1093,1326,1092,1124,346,737,951,159,1212,1234,183,940,437,568,179,240,462,]

Done.
```

L'analisi comparativa degli screenshots illustra chiaramente come ogni fase di ottimizzazione abbia contribuito a migliorare le prestazioni dell'algoritmo. Dalla versione iniziale in linguaggio C fino all'implementazione finale con strategie di precalcolo e caching avanzate, si è assistito ad una progressiva riduzione dei tempi di elaborazione, dimostrando l'efficacia delle tecniche di ottimizzazione applicate.

In conclusione, questo progetto ha dimostrato l'efficacia dell'integrazione tra tecniche di programmazione ad alto e basso livello nell'ottimizzazione dell'algoritmo Correlation Feature Selection. Attraverso un approccio metodico e iterativo, il team ha identificato e implementato miglioramenti significativi che hanno ridotto i tempi di esecuzione e ottimizzato l'utilizzo delle risorse.