ALMA MATER STUDIORUM - UNIVERSITA’ DI BOLOGNA

SEDE DI CESENA

FACOLTA’ DI SCIENZE MATEMATICHE, FISICHE E NATURALI
CORSO DI LAUREA IN SCIENZE E TECNOLOGIE INFORMATICHE

Mobilità urbana: Euristiche per la classificazione di percorsi

Relazione finale in:

Algoritmi e Strutture dati

Relatore
Prof. Luciano Margara

Presentata da
Mattia Barbaresi

Sessione II

Anno Accademico 2011/2012

Sommario

Introduzione 4
1.Concetti base 6
1.1 Machine Learning 7
1.1.1 Paradigmi di apprendimento 7
1.1.2 Approcci 9
1.2 Pattern recognition 12
1.3 Pattern matching 14
2. Strumenti 17
2.1 Applicazioni e tracciamento 17
2.2 Json 18
2.3 Mathematica 22
3. Realizzazione 24
3.1 Elaborazione del segnale 26
3.2 Euristica I 30
3.2.1 Tipologia 30
3.2.2 Implementazione 30
3.3 Euristica II 33
3.3.1 Tipologia 34
3.3.2 Implementazione 35
3.4 Euristica III 36
3.4.1 Tipologia 36
3.4.2 Implementazione 37
3.5 Costi computazionali 38
4. Risultati 42
Conclusioni e sviluppi futuri 48
Appendice 50
Bibliografia 70
Ringraziamenti 72

Introduzione

L'euristica (dal greco, "trovare") è la parte della ricerca il cui compito è di favorire l'accesso a nuovi sviluppi teorici o a scoperte empiriche. Si definisce, infatti, procedimento euristico, un metodo di approccio alla soluzione dei problemi che non segue un chiaro percorso, ma che si affida all'intuito e allo stato temporaneo delle circostanze, al fine di generare nuova conoscenza.

Nello studio di qualsiasi problema, come in questo caso la classificazione di segnali, l'euristica ha svolto un ruolo fondamentale, soprattutto nel passato, che ha portato allo sviluppo di svariate teorie le quali hanno, ad oggi, campi applicativi di grande importanza come biologia, astronomia, psicologia, cibernetica, applicazioni mediche, governative, industriali e militari.

In particolare, il riconoscimento di pattern (o *pattern recognition*) può essere considerato un processo di classificazione; il suo obiettivo è di classificare modelli (o *pattern*s), estratti dai dati analizzati, in categorie (o classi).

Un altro concetto molto importante che sarà discusso in questa tesi è il *pattern matching,* che consiste nell'individuare l'occorrenza di una certa sequenza (pattern), all'interno di una sequenza più lunga, denominata testo.

L'obiettivo di questa tesi è appunto quello di proporre alcuni procedimenti euristici per la classificazione di segnali discreti, in particolare si occupa della classificazione di percorsi riguardanti la mobilità urbana, cioè il riconoscimento dei mezzi (piedi, bici, bus, auto e treno) utilizzati per il movimento (tracciando quindi un percorso).

Esistono già numerosi algoritmi o euristiche per la classificazione di segnali che fanno uso di approcci diversi tra i quali reti neurali, reti bayesiane e macchine a vettori di supporto; la prima fase del lavoro compiuto parte proprio dallo studio della letteratura presente.

Il problema viene poi affrontato per gradi partendo con l'approccio più intuitivo, quello della prima euristica che adotta un'analisi dei parametri (come velocità, accelerazione

e varianza) del tracciato, per poi cercare nuove soluzioni ai problemi che si sono presentati nella realizzazione di questa, fino ad arrivare allo sviluppo delle altre due euristiche, più efficaci, che fanno uso di un dizionario.

Nel primo capitolo saranno quindi esposti i concetti teorici di base che servono a dare delle informazioni generali per la comprensione del lavoro svolto.

Questo capitolo sarà seguito dalla descrizione degli strumenti utilizzati per poi affrontare l’effettiva realizzazione discussa nel capitolo terzo.

I capitoli finali commenteranno i risultati delle soluzioni raggiunte e gli sviluppi futuri che possono essere realizzati da queste.

Essendo procedimenti euristici, l’obiettivo non è quello di creare soluzioni ottimizzate, ma di porre l’attenzione sugli aspetti pratici da cui prendere spunto per effettive implementazioni reali.

1.Concetti base

In questo capitolo saranno ampliati tutti quei concetti atti a dare una base necessaria per comprendere la teoria dietro gli algoritmi delle euristiche che tratteremo nei capitoli successivi.

Introduciamo dapprima il concetto di intelligenza artificiale come l'abilità di un computer di svolgere funzioni e ragionamenti tipici della mente umana.

In realtà esistono diverse definizioni contrastanti ma l'obiettivo comune dell'AI è sempre lo stesso: riuscire a mettere una macchina in condizione di imparare.

Nel suo aspetto puramente informatico, essa comprende la teoria e le tecniche per lo sviluppo di algoritmi che consentano alle macchine (tipicamente ai calcolatori) di mostrare un'abilità e/o attività intelligente, almeno in domini specifici.

Le attività e le capacità dell'Intelligenza Artificiale comprendono:

- l'apprendimento automatico (*machine learning*),
- la rappresentazione della conoscenza e il ragionamento automatico in maniera simile a quanto fatto dalla mente umana;
- la pianificazione (*planning*);
- la cooperazione tra agenti intelligenti, sia software che hardware (robot);
- l'elaborazione del linguaggio naturale (*Natural Language Processing*);
- la simulazione della visione e dell'interpretazione di immagini, come nel caso del riconoscimento facciale.

1.1 Machine Learning

"A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E".

[CIT.-Tom M. Mitchell, 1997]

Il Machine learning (o Apprendimento automatico) rappresenta una delle aree fondamentali dell’intelligenza artificiale.

Si occupa della realizzazione di sistemi e algoritmi capaci di maturare un’esperienza dall’analisi di dati empirici (presi come input) e generare, tramite questa esperienza, predizioni su nuovi dati.

Nei paragrafi seguenti sarà approfondito l’argomento del machine learning esponendo paradigmi di apprendimento e approcci degli algoritmi.

1.1.1 Paradigmi di apprendimento

Possiamo organizzare gli algoritmi tramite una tassonomia basata sul tipo di apprendimento che questi utilizzano:

- **Supervised learning,** mira a istruire un sistema informatico in modo da consentirgli di risolvere dei compiti in automatico. Nell’apprendimento supervisionato ogni esempio (*training data*)è una coppia costituita da un oggetto(tipicamente un vettore)e un valore di uscita desiderato(detto anche segnale di supervisione). Esistono due tipi di problemi per l’apprendimento supervisionato:classificazione (*pattern recognition*) e regressione (*function approximation*).

- **Unsupervised learning,** l'apprendimento non supervisionato è una tecnica che fa riferimento al problema di tentare di trovare una struttura nascosta nei dati senza etichetta. Poiché gli esempi forniti al sistema sono senza etichetta, non vi è alcun errore o segnale di supervisione per valutare una possibile soluzione. Questo distingue l'apprendimento non supervisionato

dall’apprendimento supervisionato e dall’apprendimento per rinforzo. Molti metodi impiegati si basano su metodi di estrazione dei dati utilizzati per la pre-elaborazione di questi.

- **Semi-supervised learning**, questi problemi sono caratterizzati (usualmente) dalla presenza di una piccola percentuale di dati con etichetta e da una grande percentuale di dati non etichettati. Questi problemi appaiono tipicamente in aree come: speech processing, text categorization e web categorization [1].

- **Reinforcement learning**, impara come agire dato un’osservazione del mondo. Ogni azione ha un certo impatto per l'ambiente, e l'ambiente fornisce un feedback sotto forma di premi che guida l'algoritmo di apprendimento. L’argomento dell’apprendimento per rinforzo è di grande interesse a causa del gran numero di applicazioni pratiche su cui può essere utilizzato per affrontare problemi d’intelligenza artificiale,operazioni di ricerca o ingegneria del controllo [2].

- **Transduction** o *transductive inference*, questi algoritmi di trasduzione possono essere suddivisi in due categorie: quelli che cercano di assegnare etichette ai punti discreti non etichettati, e quelli che cercano di regredire etichette continue per i punti senza etichetta. Gli algoritmi che cercano di prevedere le etichette discrete tendono a essere derivati con l'aggiunta di una supervisione parziale da parte di un algoritmo di clustering. Questi possono essere ulteriormente suddivisi in due categorie: *cluster by partitioning*[3] e *cluster by agglomerating*[4]. Gli algoritmi che cercano di prevedere etichette continue tendono a essere derivati tramite l'aggiunta di una supervisione parziale di algoritmi di *manifold learning* [5].

- **Multi-tasklearning,** tecnica che apprende un problema contemporaneamente a un insieme di altri problemi connessi, utilizzando una rappresentazione condivisa. Questo spesso porta a un modello migliore per l'attività principale,perché permette al sistema di utilizzare

caratteristiche comuni tra le attività. Questo tipo di algoritmi arricchisce il proprio insieme di assunzioni basandosi su esperienze precedenti[6].

1.1.2 Approcci

In letteratura esiste una gran quantità di approcci adottati; descriverli tutti sarebbe troppo oneroso e forviante per gli obiettivi di questa tesi. Di seguito ne riportiamo alcuni tra i più importanti, affiancati da brevi descrizioni, affinché si abbia un'idea generale sulle tecniche utilizzate.

- **Decision tree learning** [7]**,** lo scopo è di creare un modello che stimi il valore di una variabile obiettivo sulla base di diverse variabili di ingresso.

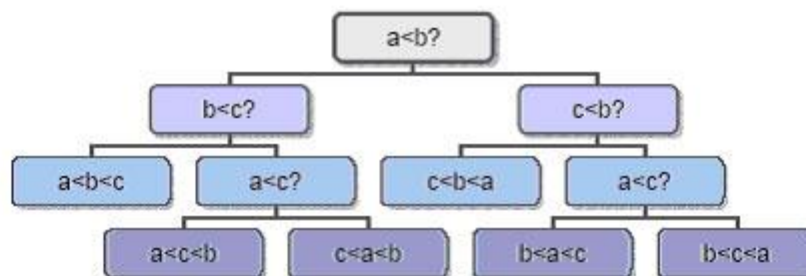


Figura 1.1.2.1 – Albero decisionale

Come in figura, ogni nodo interno rappresenta una variabile, un arco verso un nodo figlio rappresenta un possibile valore per quella proprietà e una foglia, il valore predetto per la variabile obiettivo a partire da i valori delle altre proprietà, che nell'albero è rappresentato del cammino (*path*) dal nodo radice (*root*) al nodo foglia.
Normalmente un albero di decisione è costruito utilizzando tecniche di apprendimento a partire dall'insieme dei dati iniziali (*data set*), il quale può essere diviso in due sottoinsiemi: il *training set* sulla base del quale si crea la struttura dell'albero e il *test set* che viene utilizzato per testare l'accuratezza del modello predittivo così creato.

- **Artificial neural networks** [8]**,** sono algoritmi che s'ispirano alle strutture e agli aspetti funzionali delle reti neurali biologiche. I calcoli sono strutturati in termini di un gruppo interconnesso di neuroni artificiali, il trattamento delle informazioni utilizza un approccio connessionista alla computazione. Alcuni dei campi di applicazione delle reti neurali sono: approssimazione di funzioni, Classificazione, *Data processing*, robotica.

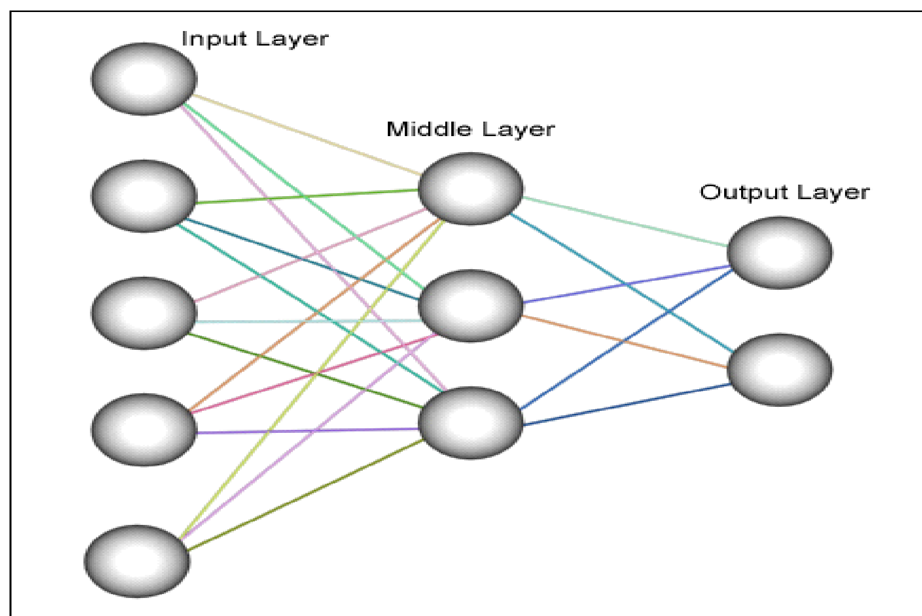


Figura 1.1.2.2 –Schematizzazione di una rete neurale

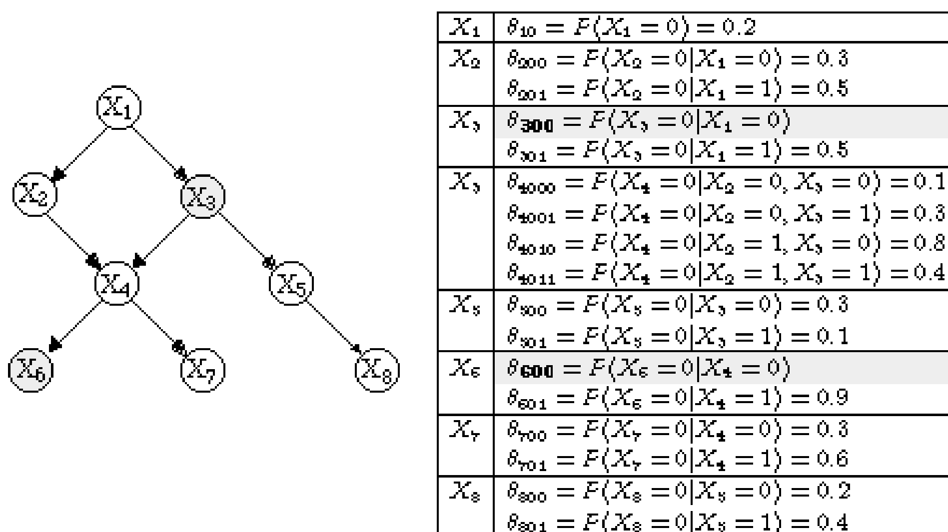
- **Genetic programming** [9]**,** è una specializzazione degli algoritmi genetici che segue le teorie evoluzionistiche di Darwin, spesso parafrasata come " la sopravvivenza del più forte". L'idea generale è che un programma può mantenere una popolazione di artefatti rappresentati utilizzando alcune opportune strutture di dati basate su computer. Elementi di tale popolazione possono quindi accoppiarsi, mutare, o altrimenti riprodursi e quindi evolversi, diretti da una misura (funzione di fitness) che valuta la qualità della popolazione rispetto alla meta del compito. La ricerca evolutiva utilizza il principio darwiniano della selezione naturale e analoghi di varie operazioni presenti in natura, tra cui cross-over(ricombinazione sessuale), mutazione,duplicazione e delezione di un gene.

- **Support vector machines** [10]**,** sono un insieme di metodi di apprendimento supervisionato per la regressione e la classificazione di pattern, sviluppati negli anni '90 da Vladimir Vapnik. I parametri caratteristici della rete sono ottenuti mediante la soluzione di un problema di programmazione quadratica convesso con vincoli di uguaglianza o di tipo box (in cui il valore del parametro deve essere mantenuto all'interno di un intervallo), che prevede un unico minimo globale.

- **Clustering** [11]**,** le tecniche di clustering si basano su misure riguardanti la somiglianza tra gli elementi. La bontà delle analisi ottenute dagli algoritmi di clustering dipende molto dalla scelta della metrica, e quindi da come è calcolata la distanza. Gli algoritmi di clustering raggruppano gli elementi sulla base della loro distanza reciproca, e quindi l'appartenenza o meno ad un insieme dipende da quanto l'elemento preso in esame è distante dall'insieme stesso.
  Le tecniche di *clustering* si possono basare principalmente su due "filosofie":
- Dal basso verso l'alto (metodi aggregativi o *Bottom-Up*):
  Prevede che inizialmente tutti gli elementi siano considerati *cluster* a sé, e poi l'algoritmo provvede ad unire i c*luster* più vicini. L'algoritmo continua ad unire elementi al *cluster* fino ad ottenere un numero prefissato di *cluster*, oppure fino a che la distanza minima tra i *cluster* non supera un certo valore, o ancora in relazione ad un determinato criterio statistico prefissato.
- Dall'alto verso il basso (metodi divisivi o *Top-Down*):
  All'inizio tutti gli elementi sono un unico cluster, e poi l'algoritmo inizia a dividere il cluster in tanti cluster di dimensioni inferiori. Il criterio che guida la divisione è naturalmente quello di ottenere gruppi sempre più omogenei. L'algoritmo procede fino a che non viene soddisfatta una regola di arresto generalmente legata al raggiungimento di un numero prefissato di *cluster*.

- **Bayesian networks** [12]**,** rappresentano modelli di probabilità congiunte tra le variabili date. Ciascuna variabile è rappresentata da un nodo in un grafo. Le dipendenze dirette tra le variabili sono rappresentate dagli archi diretti tra i

nodi corrispondenti e le probabilità condizionali per ciascuna variabile (cioè le probabilità condizionate in varie combinazioni di valori possibili per le versioni precedenti della rete) sono memorizzate in “potenzialità” (o tabelle) attaccate ai nodi dipendenti. Le informazioni sul valore osservato di una variabile si propagano attraverso la rete per aggiornare la distribuzione di probabilità sulle altre variabili che non sono direttamente osservabili. Usando la regola di Bayes, queste influenze possono essere identificate in una direzione "a ritroso", dalle variabili dipendenti ai loro predecessori.



X_1	$\theta_{10} = P(X_1 = 0) = 0.2$
X_2	$\theta_{200} = P(X_2 = 0 \mid X_1 = 0) = 0.3$ $\theta_{201} = P(X_2 = 0 \mid X_1 = 1) = 0.5$
X_3	$\theta_{300} = P(X_3 = 0 \mid X_1 = 0)$ $\theta_{301} = P(X_3 = 0 \mid X_1 = 1) = 0.5$
X_3	$\theta_{4000} = P(X_4 = 0 \mid X_2 = 0, X_3 = 0) = 0.1$ $\theta_{4001} = P(X_4 = 0 \mid X_2 = 0, X_3 = 1) = 0.3$ $\theta_{4010} = P(X_4 = 0 \mid X_2 = 1, X_3 = 0) = 0.8$ $\theta_{4011} = P(X_4 = 0 \mid X_2 = 1, X_3 = 1) = 0.4$
X_5	$\theta_{500} = P(X_5 = 0 \mid X_3 = 0) = 0.3$ $\theta_{501} = P(X_5 = 0 \mid X_3 = 1) = 0.1$
X_6	$\theta_{600} = P(X_6 = 0 \mid X_4 = 0)$ $\theta_{601} = P(X_6 = 0 \mid X_4 = 1) = 0.9$
X_7	$\theta_{700} = P(X_7 = 0 \mid X_4 = 0) = 0.3$ $\theta_{701} = P(X_7 = 0 \mid X_4 = 1) = 0.6$
X_8	$\theta_{800} = P(X_8 = 0 \mid X_5 = 0) = 0.2$ $\theta_{801} = P(X_8 = 0 \mid X_5 = 1) = 0.4$

Figura 1.1.2.3 – Esempio di rete bayesiana con relativa tabella.

1.2 Pattern recognition

Il riconoscimento è una caratteristica comune a tutti gli esseri viventi, anche se ogni creatura ha il proprio metodo. Se un umano riconosce un proprio simile attraverso la vista, dalla voce o dal modo con cui esso scrive, un cane può riconoscere un altro essere vivente attraverso l’olfatto anche a metri di distanza, cosa che per un uomo risulterebbe impossibile.

Il riconoscimento però non si limita solo ad oggetti identificabili tramite i sensi. In una conversazione potremmo subito identificare un vecchio argomento che abbiamo sentito anni prima. Tutti questi esempi sono classificabili come riconoscimento.

L’oggetto o l’entità ispezionata dal processo di riconoscimento è chiamato pattern; solitamente ci si riferisce ad un pattern come la descrizione di un’entità che vogliamo riconoscere.

Nella maggior parte dei casi il problema diventa quello di discriminazione tra diverse popolazioni. Per esempio potremmo essere interessati a suddividere un milione di persone in quattro diverse categorie: (a) alti e grassi, (b) alti e snelli, (c) bassi e grassi e (d) bassi e snelli. Vogliamo classificare ogni persona in una di queste quattro categorie. Per determinare a quale classe appartiene ogni persona dobbiamo prima trovare i tratti che caratterizzano questa classificazione. L'età di una persona, non è chiaramente un aspetto che ci possa aiutare in questo caso. Una scelta migliore è considerare la coppia (altezza, peso); in questo modo si esegue la selezione, o *features selection* delle caratteristiche per questo esempio. Raccogliere queste informazioni dal campione prende il nome di estrazione, o *features extraction.*

Nel pianificare un sistema per il pattern recognition, i.e. un sistema che sarà in grado di ricevere un pattern sconosciuto in ingresso e classificarlo in una (o più) delle diverse classi date, vogliamo chiaramente impiegare tutte le informazioni relative che sono state precedentemente raccolte. Assumiamo che siano disponibili alcuni pattern di esempio con classificazione nota. Questi pattern, con i loro attributi tipici, formano il *training set* che fornisce informazioni rilevanti su come associare i dati d'input e come prendere le decisioni (*decision making*). Tramite l'utilizzo del training set, un sistema di classificazione può apprendere vari tipi d'informazioni come parametri statistici o caratteristiche rilevanti, ecc.

Un concetto dominante, nel pattern recognition, è il *clustering* (raggruppamento). Un *cluster* è in un insieme di oggetti simili (pattern) raggruppati insieme. Il clustering consiste nella divisione dei dati in gruppi (cluster) stabilendo "i centri dei gruppi", *cluster centers*, e i loro limiti, *cluster boundaries*. Conoscere a priori il numero dei cluster e la loro locazione facilita notevolmente il lavoro; in questo caso potremmo condurre un apprendimento supervisionato (supervised learning). Se i dati non hanno alcuna caratteristica nota, si dovrà condurre un apprendimento non supervisionato (unsupervised learning). I dati in ingresso possono essere clusterizzati in modi diversi. Per esempio consideriamo le scuole di una città. Un primo raggruppamento potrebbe essere quello che riguarda la posizione geografica di queste oppure potremmo notare delle similarità tra le scuole aventi lo stesso numero di alunni, ecc.

L'obiettivo finale del pattern recognition è la classificazione.

Dalle informazioni originali abbiamo prima identificato le caratteristiche rilevanti e poi,con un *feature extractor,* abbiamo condotto una feature extraction per misurarle. Le misurazioni sono poi passate al classificatore (*classifier*) che fornisce la relativa

classificazione, i.e. stabilire a quale classe appartiene il pattern. Alcuni tipi di classificatori sono: *minimum-distance classifier*, *fuzzy classifier.*

Pattern recognition e classificazione sono tecniche usate per numerose applicazioni; tra le più importanti troviamo:

- **Applicazioni scientifiche:** astronomia, biologia, analisi dati satellitari.
- **Scienze del comportamento:** antropologia, archeologia, entomologia, biologia e botanica, psicologia, cibernetica, educazione e comunicazione.
- **Applicazioni industriali:** riconoscimento dei caratteri, riconoscimento vocale, riconoscimento immagini, esplorazione dei minerali, multimedia e animazione, progettazione giocattoli elettronici.
- **Applicazioni mediche:** esaminazioni microscopiche e dati biomedici, esaminazione radioisotopi, tomografie e esaminazione raggi-x, analisi di elettrocardiogrammi, tracciamento encefalogramma,elaborazione segnali neurobiologici.
- **Applicazioni agricole:**valutazione terreno, analisi dei campi, processi di controllo,fotografia delle risorse della terra.
- **Applicazioni governative**: predizioni del tempo, sistemi pubblici, *remote sensing*.
- **Applicazioni militari:** fotografie aeree, classificazioni e rilevazioni sonar, ATR (*Automatic Target Recognition*).

1.3 Pattern matching

Più comunemente conosciuto come *string matching*.

Algoritmi efficienti per questo problema possono migliorare significativamente la reattività dei programmi di elaborazione dei testi. Gli algoritmi di string matching sono utilizzati fra l'altro anche per la ricerca di particolari pattern nel DNA.

Il problema è di localizzare tutte le occorrenze di una stringa P di lunghezza n, chiamata pattern, in un'altra stringa T chiamata testo.

Formalmente possiamo formulare il problema nel seguente modo [13].

Supponiamo che T (1….n) sia un array di lunghezza n e che P (1…m) sia un array di lunghezza m ≤ n. Supponiamo che gli elementi di T e P siano caratteri appartenenti allo stesso alfabeto E (i.e. numeri).

Diremo che il pattern P occorre con uno spostamento *s* nel testo T (P si trova a partire dalla posizione s+1 nel testo T) se 0 ≤ s ≤ n-m e quindi T (s+1…..s+m) = P (1…m), cioè per ogni j compreso tra 1 ed m T(j)=P(j).Il problema dello string matching consiste nel trovare tutti gli spostamenti con i quali un pattern P occorre in un testo T.

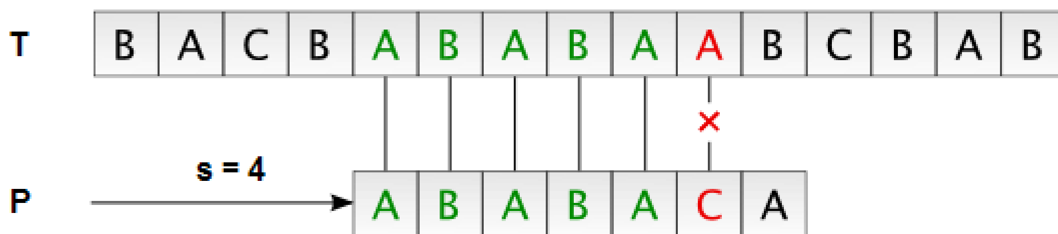


Figura 1.3.1 – String matching

Gli approcci per questo tipo di algoritmi sono essenzialmente quattro; ingenuo (*naïve*), algoritmo di Rabin-Karp, Automa a stati finiti, algoritmo KMP (Knuth-Morris-Pratt).La tabella seguente mostra i costi computazionali degli algoritmi citati.

Algoritmo	**Pre-elaboarazione**	**Matching**
Ingenuo	0	O((n-m+1)m)
Rabin-Karp	Θ(m)	O((n-m+1)m)
Automa a stati finiti	O(m|Σ|)	Θ(n)
Knuth-Morris-Pratt	Θ(m)	Θ(n)

Figura 1.3.2 – Algoritmi di String matching e tempi di elaborazione

La tabella evidenzia l'efficienza degli ultimi due algoritmi, in particolare quello di Knuth-Morris-Pratt (KMP). Per lo string matching è stata implementata una funzione che prende spunto da quest'ultimo chiamata *kmpSearch* (vedi Appendice). In questa

implementazione si è scelto di eliminare la pre-elaborazione (data la grande quantità di pattern da elaborare). In fase di esecuzione, l'algoritmo tiene traccia anche dei match per gli spostamenti successivi del pattern in modo da ottimizzare lo scorrimento della parola nel testo.

2. Strumenti

La realizzazione del progetto ha richiesto l'utilizzo di una serie di strumenti impiegati per il tracciamento, l'importazione e l'elaborazione dei segnali.

Per la cattura dei tracciati sono state implementate due applicazioni, su dispositivi *mobile,* che salvano i percorsi in file formato *json.*

Mathematica è l'ambiente di sviluppo utilizzato sia per l'implementazione sia per il test delle euristiche.

2.1 Applicazioni e tracciamento

Trattandosi di mobilità urbana, l'unico modo per costruire un database di benchmark, che rifletta le caratteristiche della viabilità in città, è quello di registrare i tracciati percorrendo le vie di questa con un dispositivo *mobile*, servendosi di apposite applicazioni create ad hoc su di esso per la cattura dei dati del GPS.

In un primo momento è stato utilizzato un iPhone 3G ma si è subito verificato un problema: con un'andatura lenta (sotto i 5 Km/h) il segnale diveniva sempre meno preciso (accuratezza = 70 metri) e le velocità non venivano rilevate dal GPS stesso.

Nella figura sottostante è graficata la velocità di un percorso a piedi in città rilevata appunto tramite l'iPhone; come si può notare su questo tracciato le velocità effettivamente rilevate dal GPS sono 5 su un totale di 500 e più campioni.

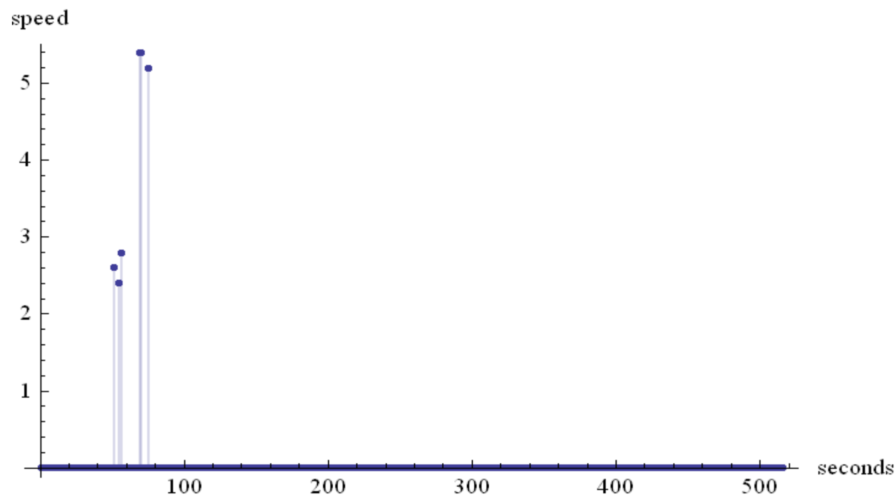


Figura 2.1.1 – Velocità a piedi rilevata con Iphone 3G

Anche se a più alte velocità l'accuratezza migliorava, il segnale del GPS veniva meno abbastanza frequentemente; in tali situazioni la velocità non era comunque rilevata.

Data la poca affidabilità del GPS dell'iPhone, si è scelto di cambiare dispositivo, tutto lo studio, quindi, si basa su tracciati catturati tramite un Samsung Galaxy SII.

Le due applicazioni create per il tracciamento, per i dispositivi sopra citati, differiscono solo in grafica; entrambe all'atto pratico scrivono in output un file json, contenente il tracciato salvato, descritto nel paragrafo seguente.

2.2 Json

JSON (JavaScript Object Notation) è un semplice formato per lo scambio di dati. Per le persone è facile da leggere e scrivere, mentre per le macchine risulta facile da generare e analizzarne la sintassi. Si basa su un sottoinsieme del linguaggio di programmazione JavaScript[14].

JSON è un formato di testo completamente indipendente dal linguaggio di programmazione, ma utilizza convenzioni conosciute dai programmatori di linguaggi della famiglia del C, come C, C++, C#, Java, JavaScript, Perl, Python, e molti altri. Questa caratteristica fa di JSON un linguaggio ideale per lo scambio di dati.

JSON è basato su due strutture:

- Un insieme di coppie nome/valore. In diversi linguaggi, questo è realizzato come un oggetto, un record, uno struct, un dizionario, una tabella hash, un elenco di chiavi o un array associativo.
- Un elenco ordinato di valori. Nella maggior parte dei linguaggi questo si realizza con un array, un vettore, un elenco o una sequenza.

Queste sono strutture di dati universali. Virtualmente tutti i linguaggi di programmazione moderni li supportano in entrambe le forme. E' sensato che un formato di dati che è interscambiabile con linguaggi di programmazione debba essere basato su queste strutture.

In JSON, codeste strutture assumono queste forme:

Un *oggetto* è una serie non ordinata di nomi/valori. Un oggetto inizia con { (parentesi graffa sinistra) e finisce con } (parentesi graffa destra). Ogni nome è seguito da : (due punti) e la coppia di nome/valore sono separata da , (virgola).

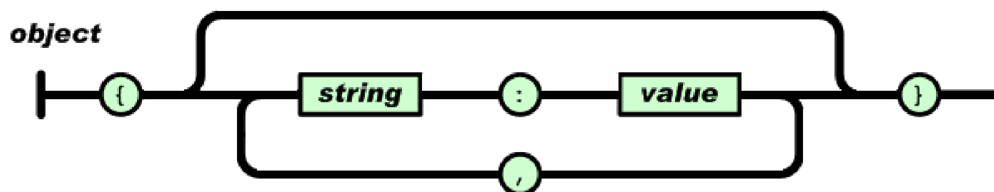


Figura 2.2.1

Un *array* è una raccolta ordinata di valori. Un array comincia con [(parentesi quadra sinistra) e finisce con] (parentesi quadra destra). I valori sono separati da , (virgola).

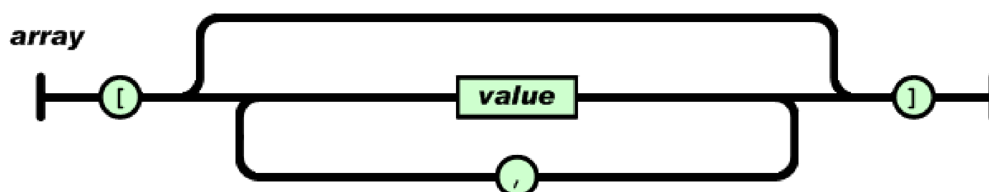


Figura 2.2.2

Un *valore* può essere una stringa tra virgolette, un numero, o vero o falso o nullo, o un oggetto o un array. Queste strutture possono essere annidate.

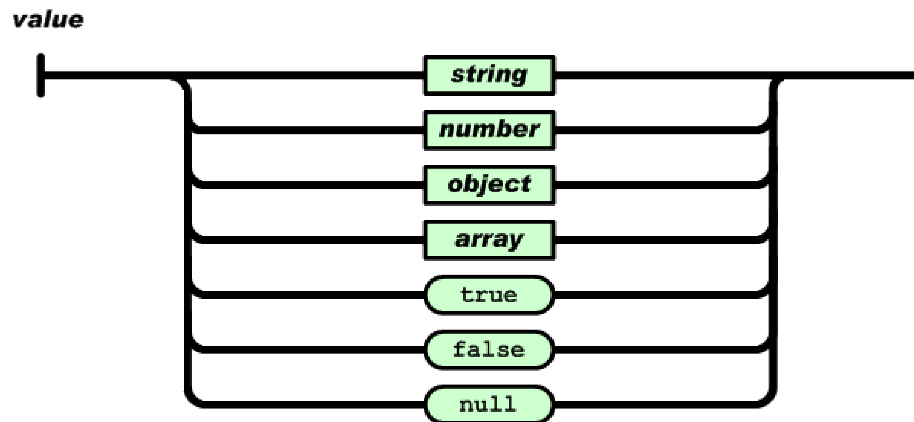


Figura 2.2.3

Una *stringa* è una raccolta di zero o più caratteri Unicode, tra virgolette; per le sequenze di escape utilizza la barra rovesciata. Un singolo carattere è rappresentato come una stringa di caratteri di lunghezza uno. È molto simile ad una stringa C o Java.

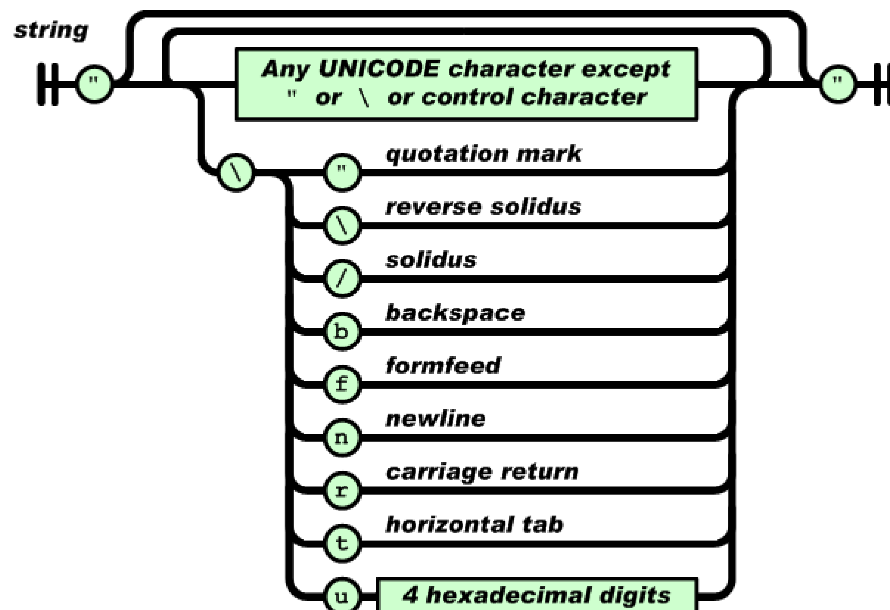


Figura 2.2.4

Un *numero* è molto simile al corrispettivo in C o Java, a parte il fatto che i formati ottali e esadecimali non sono utilizzati.

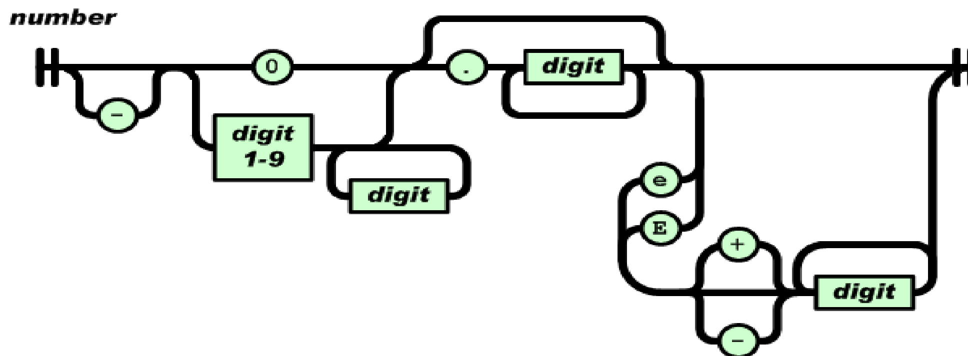


Figura 2.2.5

I caratteri di spaziatura possono essere inseriti in mezzo a qualsiasi coppia di token.

Quanto segue, riporta un esempio di testo in formato json utilizzato. Nelle parentesi graffe sono racchiusi i parametri relativi a una singola posizione rilevata dal GPS. Il *mezzo* è un'etichetta aggiunta dall'applicazione.


```
[
{
"mezzo": "Auto",
"latitudine":"0",
"longitudine": "12.248990759253502",
"altitudine": "96.5999755859375",
"speed": "49.539597",
"accuratezza": "10.0",
"data": "1352832465000"
}
,
{
"mezzo": "Auto",
"latitudine":"0",
"longitudine": "12.248862432315946",
"altitudine": "96.20001220703125",
"speed": "48.638878",
"accuratezza": "10.0",
"data": "1352832466000"
}
]
```

2.3 Mathematica

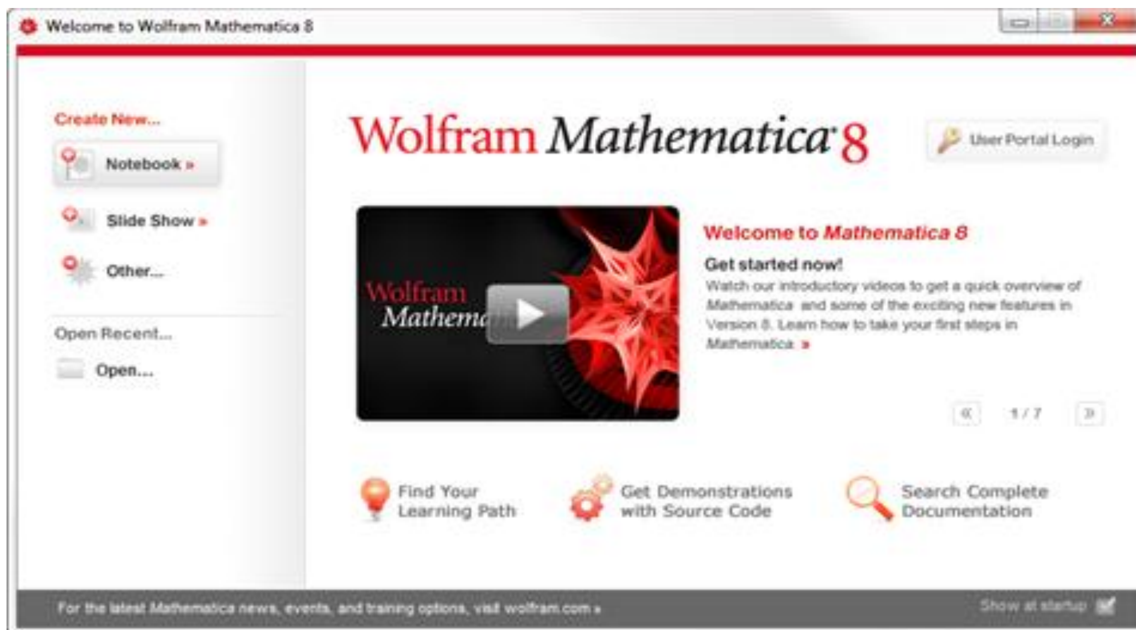


Figura 2.3.1 – Welcome screen di Mathematica 8

Almost any workflow involves computing results, and that's what Mathematica does—from building a hedge fund trading website or publishing interactive engineering textbooks to developing embedded image recognition algorithms or teaching calculus[15].

Mathematica è un ambiente di calcolo simbolico e numerico multipiattaforma, è anche un potente linguaggio di programmazione interpretato,ideato da **Stephen Wolfram** nel 1986, la prima versione fu rilasciata nel 1988. La versione attuale è la 9.0.0, rilasciata nel 2012.

Il linguaggio di programmazione di Mathematica è basato sulla riscrittura di espressioni (*term-rewriting*)[16] e supporta svariati paradigmi di programmazione, tra cui la programmazione funzionale, la programmazione logica, la programmazione basata sul riconoscimento di schemi (*pattern-matching*)[17] e sulle regole di sostituzione (*rule-based*)[18], nonché la più tradizionale programmazione procedurale. Mathematica è realizzato principalmente in C e C++, ma gran parte delle numerose librerie fornite con il

programma sono scritte nel linguaggio proprietario di Mathematica, che può essere utilizzato per espandere ulteriormente le funzionalità del sistema.

In Mathematica, il linguaggio di base viene interpretato da un *kernel* che esegue l'elaborazione vera e propria; i risultati vengono quindi comunicati ad uno specifico *front end* tra quelli disponibili. La comunicazione tra il kernel e questi ultimi (o qualsiasi altro client, ad esempio programmi scritti dall'utente) utilizza il protocollo *MathLink*, spesso attraverso una rete. È possibile che vari processi front-end si connettano allo stesso kernel, e che uno stesso front-end sia connesso a kernel differenti.

Di norma, il front-end è rappresentato da un documento di testo interattivo, il **notebook** (blocco per gli appunti), che è in grado di visualizzare ed interpretare la notazione matematica bidimensionale in formato WYSIWYG(acronimo inglese che sta per *What You See Is What You Get,*"quello che vedi è quello che è") e incorpora i risultati dell'elaborazione sotto forma di testo, formule, grafici e suoni. I notebook sono file di testo in formato ASCII con estensione**.nb** che possono essere passati da una piattaforma all'altra.

3. Realizzazione

In questo capitolo saranno descritti i particolari degli algoritmi in questione: le assunzioni fatte, le scelte condotte e tutto quello che riguarda l'aspetto realizzativo per questi algoritmi. Lo scopo non è quello di creare algoritmi ottimizzati ed efficienti al massimo, bensì quello di dare spunto per vere e proprie implementazioni future.

I parametri forniti dal GPS sono vari: altitudine, longitudine, data, direzione, latitudine, accuratezza del segnale e velocità. La prima scelta è stata quella di considerare solo quest'ultimo valore. In un primo momento parrebbe una scelta insensata ma un ragionamento approfondito evidenzia la stretta dipendenza dei parametri rispetto alla velocità. Se un'auto viaggia alla stessa velocità di un uomo che passeggia, entrambi percorrono lo stesso spazio. Anche l'accelerazione è calcolata come la variazione di velocità in un determinato istante.

Un altro parametro importante è la data. Il GPS fornisce fino al secondo in cui è stata catturata la posizione. In termini di velocità, è un dato assai importante perché permette il calcolo dell'accelerazione,altrimenti impossibile. Le applicazioni create per il tracciamento hanno un timer che scandisce il tempo di cattura (ovvero il salvataggio) della posizione: il *rate*. Per semplicità questo rate è impostato ad 1 (ogni secondo), in questo modo non è più necessario tener conto della variazione del tempo da un rilevamento all'altro (perché di un secondo). Inoltre, se si vuole aumentare il rate, basta leggere i salvataggi in modo appropriato (i.e., se voglio un rate uguale a 2, ovvero ogni due secondi, basta leggere un salvataggio si e uno no, ecc..). Questo rate è molto importante perché è un aspetto considerato nello studio di queste euristiche di cui parleremo nei capitoli successivi.

Per questi motivi si è scelto di attingere, dai parametri che il GPS fornisce,esclusivamente la velocità.

Nella figura sotto è rappresentato, graficamente, un percorso catturato.L'asse delle ascisse rappresenta il tempo (in secondi) mentre l'asse delle ordinate rappresenta la velocita (in Km/h).

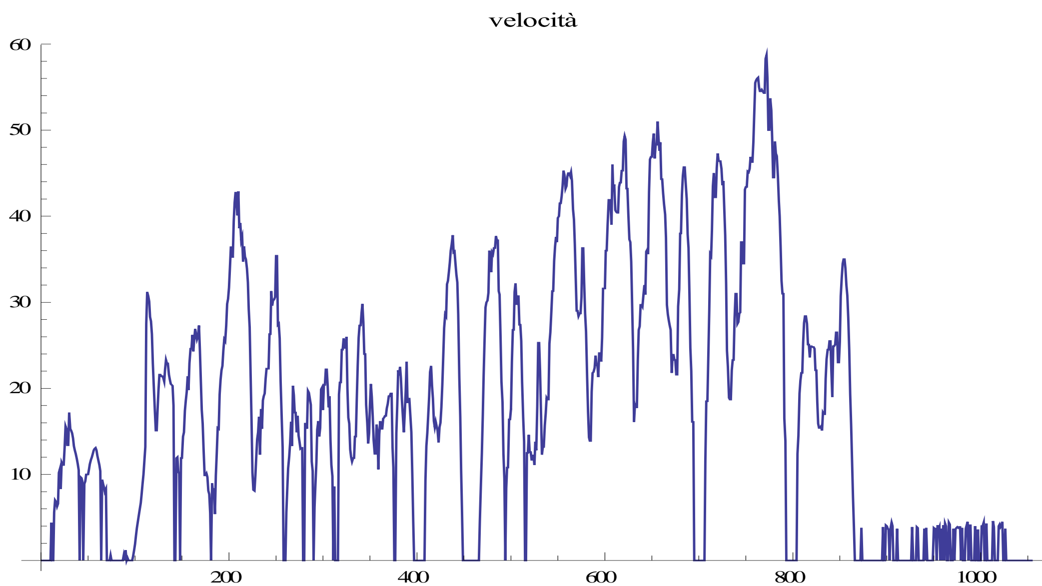


Figura 3.1 – Tracciato catturato in autobus e poi a piedi.

Un aspetto comune a tutte le implementazioni è la funzione *RangeV[speed_,indice_,interv_]* (vedi Appendice). Questa funzione è usata per spezzare il tracciato e creare degli intervalli, che poi saranno ulteriormente divisi in pattern per essere classificati. Adotta assunzioni sul tempo d'inattività (secondi in cui la velocità è pari a zero) per la suddivisione di questi intervalli ("*Per poter cambiare mezzo devi fermarti!*"). Un intervallo, in questo senso, è una parte del tracciato che ha inizio e fine con velocità pari a zero.

In generale, una volta che l'algoritmo è terminato, non tutti gli intervalli potrebbero essere etichettati. Questo perché potrebbero essere presenti degli intervalli, in cui la velocità è costantemente nulla, non considerati dalla funzione RangeV. Ovviamente se non ci si sta muovendo, è impossibile determinare il mezzo ma si potrebbero fare delle assunzioni del tipo *"se prima ero sul mezzo M e dopo sono sullo stesso mezzo M, allora anche adesso sarò sul mezzo M"*. Questo è il compito di un'altra funzione comune a tutte le euristiche: *RiempiZeri[vettor_]*(vedi Appendice). Nella maggior parte dei casi questa funzione aumenta il numero di etichettamenti esatti forniti dall'algoritmo.

Lo studio è stato condotto per gradi; la prima euristica descrive l'approccio più semplice e intuitivo. Realizzata questa, si è potuta avere una visione più completa e pratica del problema. Quest'algoritmo, ovviamente, non è il più ingegnoso ma è stato fondamentale per la realizzazione dei successivi.

Le altre due implementazioni fanno uso di un dizionario per la classificazione dei percorsi.

3.1 Elaborazione del segnale

L'elaborazione del segnale svolge un ruolo importante nello studio che è stato condotto. Il tracciato così come viene catturato presenta del rumore, cioè porta con se delle perturbazioni dovute dall'accuratezza o dall'assenza momentanea del segnale del GPS. Di conseguenza, è necessaria una rettifica che raffini il percorso ma, allo stesso tempo, non provochi la perdita di informazioni. Queste modifiche sono state realizzate per mezzo di medie mobili o tramite una scelta appropriata del rate, ovviamente si deve fare attenzione perché un termine troppo lungo per la media mobile o un valore troppo alto del rate causano importanti perdite di informazioni come l'accelerazione o la varianza del segnale. In Figura 3.1.1 è mostrato un campionamento d'intervallo, preso tra i 200 e i 400 secondi, di un tracciato percorso a piedi a cui non è stata sottoposta alcuna elaborazione; questo sarà preso come modello per graficare le modifiche fatte al segnale.

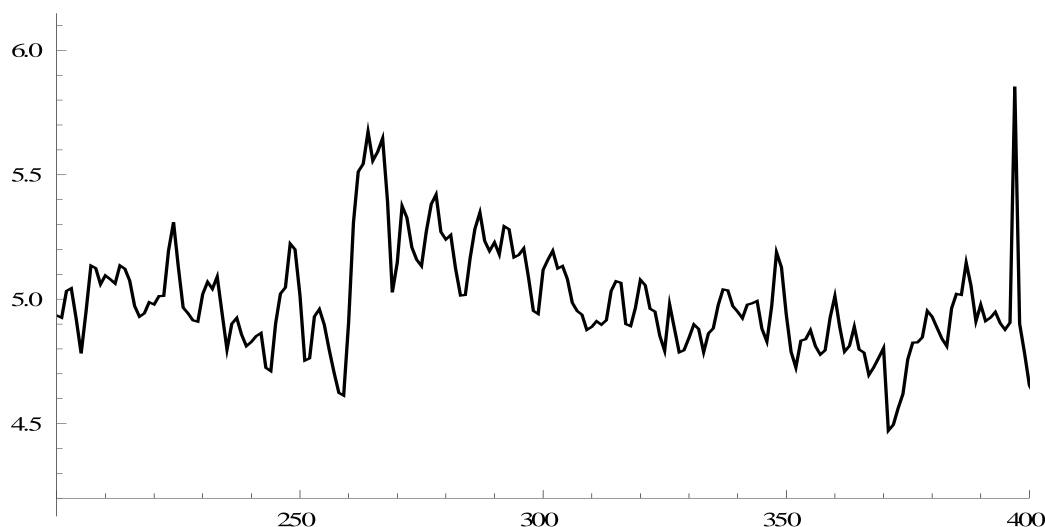


Figura 3.1.1 – Segmento di un tracciato a piedi

Focalizziamo per un attimo l'attenzione sul picco nella parte finale del tracciato; se consideriamo che il tracciato sia stato catturato mantenendo un'andatura abbastanza costante, in un secondo è stata rilevata una variazione di velocità anomala per un uomo che passeggia.

Questa variazione è causata proprio dalla scarsa accuratezza del segnale del GPS. In questi casi, l'uso di una media mobile appropriata potrebbe risolvere il problema.

In Figura 3.1.2 mostriamo lo stesso tracciato della figura precedente a cui è stata applicata una media mobile con termine uguale a cinque.

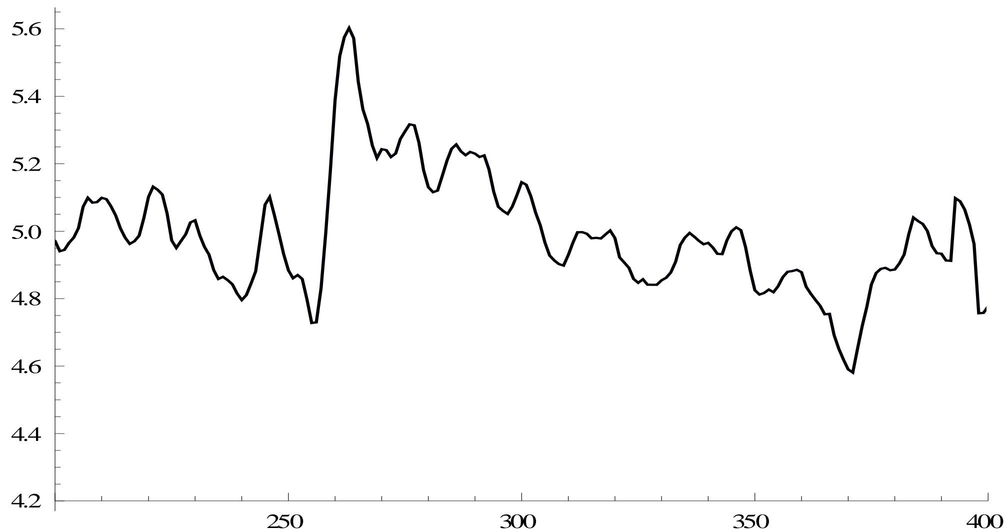


Figura 3.1.2 – Media mobile con termine uguale a 5

Subito si nota come la media mobile ha reso più pulito il segnale, in particolare è sparito quel picco anomalo che si presentava nel grafico di partenza.

L'utilizzo di queste medie è stato rilevante soprattutto in quegli algoritmi che utilizzano il tracciato per il matching nella ricerca all'interno del dizionario. La prima euristica non è molto influenzata in quanto adotta un'analisi dei parametri del segnale e non del segnale stesso.

Come detto in precedenza, l'utilizzo di medie mobili, potrebbe causare una perdita d'informazioni. Queste tendono ad appiattire il segnale; più si allunga il termine della media più il segnale sarà arrotondato.

Nel grafico sottostante (Figura 3.1.3) è mostrato lo stesso esempio cui è stata applicata però una media mobile con termine uguale a dieci.

Anche qui il segnale risulta privo di rumore ma si cominciano a notare gli effetti di una media mobile con termine troppo elevato, cioè si comincia a denotare una perdita di informazioni dovuta all'eccessivo arrotondamento che questa apporta.

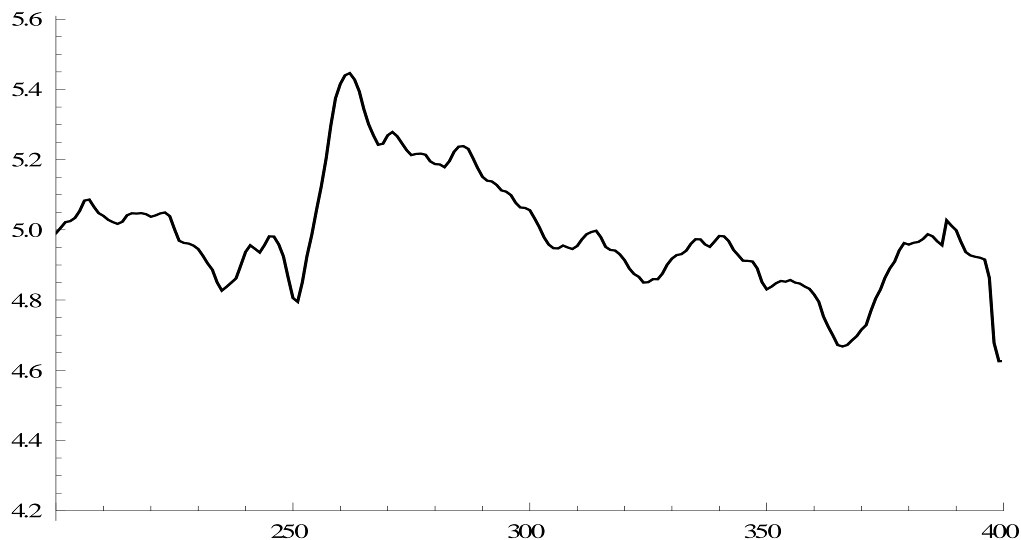


Figura 3.1.3 - Media mobile con termine uguale a 10

Come si vede in figura (Figura 3.1.3), con l'allungarsi del termine della media, le repentine variazioni dovute al rumore scompaiono ma scompaiono anche quelle variazioni naturali dell'andatura.

Un altro modo che potrebbe risolvere il problema del rumore è la scelta appropriata del rate di campionamento. In questo modo però la perdita di segnale è certa poiché si tolgono al segnale dei punti di rilevamento. Il segnale preso come modello in Figura 3.1.1 ha un rate pari a uno, cioè il campionamento è stato eseguito ogni secondo. Questo significa che, ogni secondo, è stata catturata una rilevazione del segnale da parte del GPS.

Di seguito, invece, mostriamo lo stesso tracciato cui è stato applicato un rate pari a due.

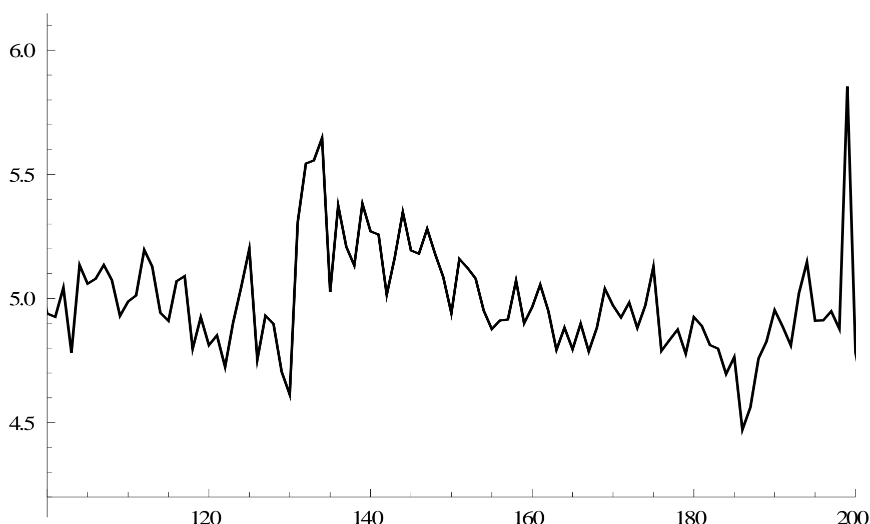


Figura 3.1.4 – Tracciato a piedi con rate uguale a due

La differenza col tracciato preso con un rate uguale a uno è che, utilizzando un rate pari a due si eliminano quelle oscillazioni vicine al segnale ma non si pone rimedio ai picchi come quello evidenziato in precedenza.

Ovviamente con un rate pari a due la lunghezza del tracciato è dimezzata; l'intervallo preso in considerazione ha una lunghezza di 100 secondi con l'intervallo campionato tra i 100 e i 200 secondi.

Aumentando il rate ancora di un punto (rate = 3), si evidenzia maggiormente il comportamento appena citato. In questo caso, l'intervallo è ridotto di un terzo rispetto a quello del modello.

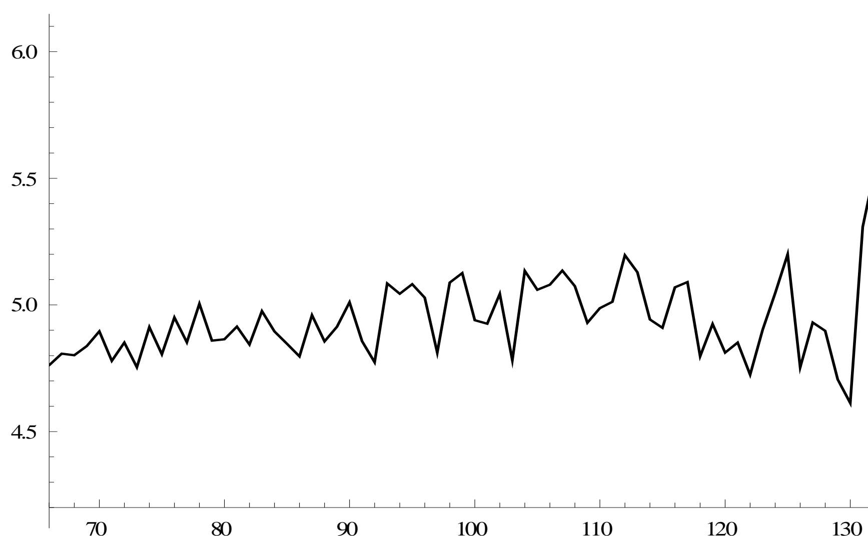


Figura 3.1.5 - Tracciato a piedi con rate uguale a tre

In Figura 3.1.5 è evidente come un rate troppo alto provochi una distorsione eccessiva del segnale in generale, anche se, poi, è riuscito a livellare il picco nel finale.

Man mano che aumentiamo il rate si perdono informazioni ma si diminuiscono drasticamente le lunghezze dei tracciati.

Da questo punto di vista, il rate abbassa notevolmente il costo di computazione perché permette una riduzione dei dati da elaborare, d'altro canto la perdita di informazioni rende più difficoltosa una classificazione esatta.

Per quanto riguarda il primo algoritmo, non sono state applicate medie mobili perché queste incidevano troppo su parametri come l'accelerazione, si è preferito quindi porre l'attenzione sullo studio del rate. Per gli algoritmi che fanno uso del dizionario, si è utilizzata una media mobile con termine pari a dieci per facilitare il matching dei segnali.

3.2 Euristica I

Un primo ragionamento ha condotto alla classificazione del segnale attraverso l'analisi dei suoi parametri; da quelli propri del segnale come: velocità massima e media (calcolata anche con l'esclusione dei tempi inattivi) o accelerazione massima e media, a quelli riguardanti i dati in generale come la varianza.

Tramite un'indagine sui percorsi catturati, si ottengono questi valori usati poi per decidere l'appartenenza di un percorso ad una classe piuttosto che ad un'altra.

Per dare un esempio: se la mia velocità media non supera i 10 Km/h, potrei essere a piedi, in bici o addirittura in macchina (magari in coda) ma sicuramente non sarò in treno (che non soffre il traffico), al contrario se la mia velocità massima (o media) supera i 100 Km/h potrei essere in auto o in treno ma sicuramente non starò camminando o andando in bicicletta.

Il problema è che in questo modo possiamo solo definire dei limiti superiori per le classi e quindi si verranno a creare delle situazioni in cui, i segnali di un autobus o di una macchina, per esempio, vengono "descritti" allo stesso modo dai parametri che si analizzano e questo creerà degli etichettamenti sbagliati.

3.2.1 Tipologia

La classificazione è fatta tramite un albero decisionale (*decision tree*) che analizza velocità massima e media, accelerazione e scarto quadratico medio.

La decisione è condotta tramite la valutazione di questi parametri fino ad arrivare alle foglie che rappresentano l'etichetta (*label*) da assegnare al pattern.

3.2.2 Implementazione

Il comportamento di questa euristica è molto semplice. L'algoritmo riceve (in input) un tracciato, da questo estrae il vettore delle velocità (per un esempio grafico vedi Figura 3.1). A questo punto cerca all'interno di questo vettore la velocità massima raggiunta (in tutto il tracciato) e, tramite la funzione *RangeV,* calcola l'intervallo cui quella velocità appartiene.

Per calcolare il range, si poteva scegliere un punto di partenza qualsiasi; se il valore scelto è compreso in un intervallo d'inattività, verrà calcolato un range inattivo dove l'algoritmo semplicemente non porrà etichetta.

Si è scelto di partire dalla velocità massima presente nel tracciato perché da questa si calcola un intervallo "sicuro", cioè si calcola un intervallo che, di solito, è uno tra i più ampi quindi che porta ad una classificazione più affidabile.

Una volta ottenuto l'intervallo, tramite la funzione *Mezzo[pezzo_]* (vedi Appendice), calcola la velocità media, la velocità massima, il sigma (scarto quadratico medio) e l'accelerazione massima.

Velocità media e sigma sono calcolati escludendo i valori nulli delle velocità nel vettore, questo per avere dei parametri più veritieri delle andature; pensiamo ad un autobus la cui velocità media sarebbe molto inferiore a quella di crociera se considerassimo anche i tempi in cui sosta nelle fermate.

Una volta ottenuti questi valori, l'algoritmo (tramite una serie di verifiche) li utilizza per dare una classificazione del mezzo.

In un primo momento, l'algoritmo era concepito in modo tale che scelta una classe di appartenenza questa escludesse le altre.

```
If[(media < 10 && sigma < 3 && acc < 5 ),
 mezzo = "P" ;
 ,
 If[(sigma < 20 && acc > 5 && media > 10),
   mezzo = "A" ;
   ,
   If[( sigma > 10 && acc > 8),
     mezzo = "M" ;
     ,
     If[(2 <= sigma <= 7 && media > 5 && maxV < 35 ),
       mezzo = "B";
       ,
       If[(maxV > 120 && 1 < acc < 8 && media > 85),
         mezzo = "T"];
       ];
     ];
   ];
 ];
```

Questo però era solo un prototipo che non poteva funzionare data la grande varietà di dati. Le regole stabilite erano troppo rigide e selettive.

A questo punto si è deciso di adottare un albero di decisione che permettesse la valutazione di più casistiche, capace di adattarsi meglio alle situazioni. I valori utilizzati per i confronti sono valori statistici calcolati sul training set.Di seguito riportiamo la schematizzazione del decision tree usato.

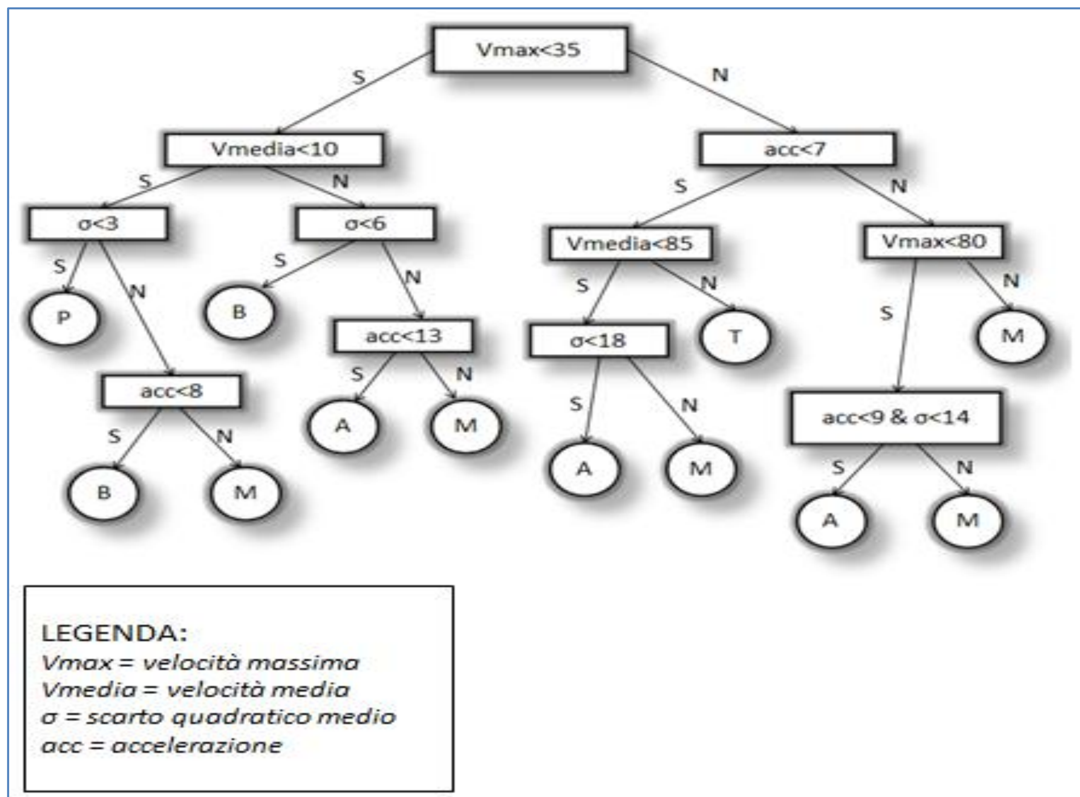


Figura 3.2.2.1 – Decision tree utilizzato e relativa legenda

Con questo metodo, l'algoritmo è in grado di prendere decisioni diverse in base alla "situazione" in cui si trova. Questo permette una maggiore adattabilità rispetto alla molteplicità dei dati. Ulteriori studi potrebbero essere condotti per trovare alberi decisionali ancora più efficaci.

Quanto detto descrive il comportamento della prima euristica, di seguito ne riportiamo il codice.

```
E1[tracciato_]:=

 Module[{i,max,speed,pezzo,vettore,mezzo,tab},
  (

   tab=Flatten[CreaTabellaTracciato[tracciato,{"speed"}]];
   speed=Flatten[Table[tab[[i]],{i,1,Length[tab],rate}]];
   vettore=speed;

   While[Max[speed]>0.2,
    max=MaxV[speed];
    pezzo=RangeV[speed,max[[2]]];
    mezzo=Mezzo[pezzo];

    For[i=1,i<= Length[pezzo],i++,
     speed[[pezzo[[i,1]]]]=0.;
     vettore[[pezzo[[i,1]]]]=mezzo;
     ];
    ];
   vettore=RiempiZeri[vettore];
   Return[vettore]
   )
  ];
```

La velocità di esecuzione è il pregio di questa euristica capace di classificare un tracciato in tempo lineare.

Tutte le funzioni utilizzate con relativo codice sono riportate in Appendice.

3.3 Euristica II

La prima euristica evidenzia il problema oggetto dello studio in questione cioè la difficoltà di etichettare pezzi di tracciati simili tra loro ma catturati con mezzi diversi.Utilizzando un'analisi dei parametri si generalizzano troppo le caratteristiche dei segnali incorrendo, di conseguenza, nell'errore appena citato.

Quest'euristica cerca di ovviare al problema con un approccio del tutto diverso. L'idea è di avere un dizionario di percorsi suddiviso in base al mezzo con cui è stato catturato

il tracciato.Presi i pattern di un intervallo da un tracciato, questi vengono confrontati con i percorsi presenti nel dizionario e la cartella (che contiene i percorsi relativi ad un solo mezzo) che riceve maggior corrispondenza determina il mezzo per quell'intervallo.

Questo metodo dovrebbe offrire, oltre che una maggior precisione, una maggiore adattabilità a tutti quei casi limite sopra citati.

3.3.1 Tipologia

In questo caso si può parlare di supervised learning, ovvero apprendimento supervisionato.Questo perché l'utilizzo di un dizionario conferisce esperienza all'algoritmo: se il dizionario è vuoto, non sarà possibile etichettare nessun tipo di percorso maaggiungendo dei tracciati con etichetta (ecco perché supervisionato) l'algoritmo acquisterà esperienza per poterne classificare di nuovi. Questo non vuol dire necessariamente che più tracciati si aggiungono migliori saranno le prestazioni dell'algoritmo; superata una certa soglia, i tracciati aggiunti saranno ridondanti e provocheranno solo un aumento del tempo richiesto per l'elaborazione.

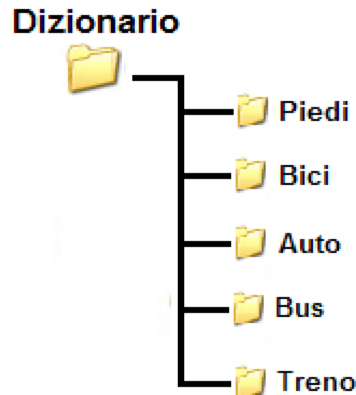


Figura 3.3.1.1 – Struttura dizionario

In figura è mostrata la struttura del dizionario. All'interno di ogni cartella sono memorizzati i relativi tracciati di appartenenza Il nome della cartella rappresenta anche la classe utilizzata per l'etichettamento.

3.3.2 Implementazione

Anche quest'algoritmo riceve (in input) un tracciato e da qui ne estrae le velocità. Dato l'alto numero di cifre decimali per questi valori (fino a otto) è prevista un'elaborazione del segnale, tramite media mobile e arrotondamento, peragevolare il match tra i segnali.

Una volta conclusa questa elaborazione, tramite la funzione *Punti[speed_,offset_]* (vedi Appendice) viene calcolato un vettore che contiene i minimi e massimi relativi del segnale.Da questo vettore, per ogni punto viene calcolato il relativo intervallo di appartenenza che viene a sua volta suddiviso in pattern utilizzati per il matching con i segnali presenti nel dizionario.

Per agevolare l'esecuzione, la funzione *Scelta[speed_]* (vedi Appendice), in maniera simile alla prima euristica, sceglie in quali cartelle effettuare la ricerca.Una volta stabilite le cartelle, l'algoritmo legge i file (tramite le funzioni *LeggiCartellaDizionario[cartella_]* e *LeggiVoceDizionario[nome_, cartella_],* vedi Appendice), ne elabora il segnale e li confronta uno ad uno con il pattern.Il confronto tra il pattern e tracciato del dizionario, il pattern matching, è implementato dalla funzione kmpSearch di cui abbiamo già parlato.

Una variante di questa euristica è stata implementata modificando la funzione di matching.In questo caso la funzione *d[p1_, p2_]* (vedi Appendice) calcola una sorta di distanza tra i due segnali come la media delle distanze tra le coppie di punti (uno del testo e l'altro del pattern). Se questa distanza è pari a zero i due segnali coincidono.

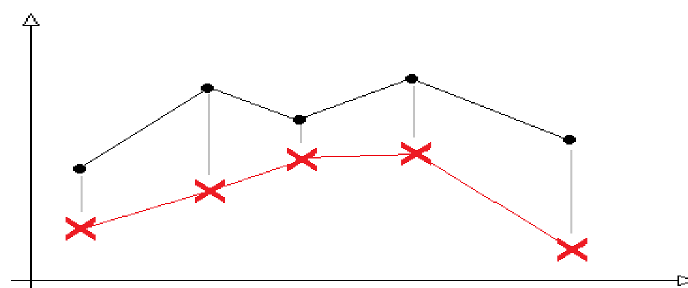


Figura 3.3.2.1 – Distanza tra segnali

Con questa variante però non si sono visti miglioramenti sostanziali.

Una volta finito il matching viene scelta la cartella che ha più congruenze con il pattern visionato e viene fatto l'etichettamento. Anche in questo caso, la funzione RiempiZeri raffina (di solito) la classificazione appena compiuta.

Per motivi di spazio il codice di questa euristica è riportato solo in fondo (vedi E2 in Appendice).

Questa euristica è sicuramente più robusta della prima ma è anche più lenta, infatti, per ogni pattern deve andare ad ispezionare l'intero dizionario (o quasi).

3.4 Euristica III

Con questa euristica si è cercato di creare un algoritmo abbastanza efficace quanto la seconda euristica ma più efficiente, cioè più veloce in fase di elaborazione. La lunga elaborazione della seconda euristica è dovuta al fatto che per ogni pattern, l'algoritmo deve avviare una ricerca su tutti i percorsi memorizzati nel dizionario.

Nella progettazione della terza euristica si è pensato di dividere la gestione dei segnali del dizionario dalla classificazione vera e propria. È stata concepita una funzione che estrapola dal dizionario i pattern più caratteristicidi ogni cartella (quindi di ogni mezzo). In fase di apprendimento l'algoritmo ricerca i pattern più popolari e stila una classifica per ogni classe che viene salvata in un file. A questo punto, nella fase di ricerca, per classificare un pattern basterà confrontarlo con i pattern presenti in questa classifica per decidere che etichetta assegnargli.

Ovviamente la fase di apprendimento è molto lunga e laboriosa ma sarà eseguita solo una volta, o comunque ogni volta che un percorso nuovo è inserito nel dizionario e ciò non capita di frequente, in compenso la classificazione risulterà più veloce e meno faticosa.

3.4.1 Tipologia

Esattamente come nella seconda euristica e nella maggior parte dei problemi di classificazione simili siamo in presenza di un apprendimento supervisionato. Anche quest'algoritmo, per la fase di apprendimento, adotta lo stesso dizionario utilizzato dal secondo approccio.

3.4.2 Implementazione

In questo paragrafo sarà descritto l'aspetto implementativo della terza euristica. La fase di valutazione del training set è implementata attraverso la funzione *Apprendimento[]* (vedi Appendice) ed è separata dall'algoritmo di classificazione. Questa funzione permette di scegliere il rate, la lunghezza dei pattern e il numero di pattern calcolati per ogni classe. L'apprendimento termina scrivendo in un file la classifica dei pattern caratteristici. L'algoritmo di classificazione, in fase di run-time andrà a leggere questo file per ottenere i pattern necessari per la classificazione.

Fase 1: Apprendimento

Questa è la fase più laboriosa dell'euristica poiché deve scansionare l'intero dizionario alla ricerca dei pattern più ricorrenti. Per ogni cartella, legge tutti i file e ne elabora il segnale con la stessa funzione utilizzata nella seconda euristica. Una volta elaborato il segnale, prende ogni tracciato e lo spezza in pattern di lunghezza prefissata; per ognuno di questi pattern avvia una ricerca su tutti i file della cartella corrispondente e ne conta il numero di occorrenze trovate.Grazie alla funzione *Score[n_,pattern_,classifica_,occ_,len_]*(vedi Appendice)è mantenuta una classifica per ogni cartella dei suoi pattern caratteristici (più frequenti).Una volta terminato l'apprendimento, il sistema genererà un file chiamato "*Patterns.json*" contenente un vettore di cinque posizioni (una per ogni classe) dove per ognuna di queste è salvata la lista di questi pattern. Questo file verrà letto dall'algoritmo di classificazione.

3.4.2.1 Fase 2: Classificazione

L'implementazione dell'algoritmo che sta per essere descritto fa riferimento alla funzione *E3[tracciato_]* (vedi Appendice).

Come per la seconda euristica si riceve in input il tracciato da cui si estrae la velocità che viene poi elaborata. A questo punto viene letto il file "Patterns.json" e viene creato il vettore che contiene i pattern calcolati dalla funzione di apprendimento. Questa volta l'intervallo calcolato da RangeV non è ulteriormente suddiviso in parti perché, in questo caso, utilizziamo i pattern calcolati per classificare l'intervallo; non devo quindi calcolarei pattern a partire dall'intervallo, come succede nella seconda euristica. Attraverso kmpSearch sono contate le occorrenze totali per ogni classe e la classe con il numero di *hit* più alto viene eletta per la classificazione dell'intervallo.

3.5 Costi computazionali

Per discutere l'effettiva efficienza degli algoritmi proposti è necessariocalcolarne il tempo di esecuzione. Riporteremo quindi lo pseudocodice delle tre euristiche seguito dall'analisi del costo computazionale per ognuna di esse. Per ogni istruzione riporteremo il costo e il numero di volte che questa viene eseguita.

Il tempo di esecuzione dell'algoritmo è la somma dei tempi di esecuzione per ogni istruzione eseguita; un'istruzione che richiede *p* passi e viene eseguita *n* volte contribuirà con costo$p*n$ al tempo di esecuzione totale.

Per semplificare i calcoli, però, dobbiamo fare delle supposizioni che ci aiuteranno a generalizzare i casi possibili.

Assumeremo che:

- Tutte le operazioni aritmetiche e gli assegnamenti abbiano costo pari a 1,
- I tracciati abbiano tutti una lunghezza media*L,*
- Ogni tracciato venga suddiviso in un numero di range medio *nr,*
- I range abbiano tutti lunghezza media *lr,*
- Ogni cartella del dizionario abbia un numero medio difile *nf,*
- I file del dizionario abbiano tutti lunghezza media *lf.*

	E1	**costo**	**numero di volte**
1	**While** Max[speed] > 0	*1*	*nr*
2	**do** max←Max[speed]	*L*	*nr*
3	pezzo←RangeV[speed,max]	*lr*	*nr*
4	mezzo←Mezzo[pezzo]	*4lr+4*	*nr*
5	**for** i ← 1 to Length[pezzo]	*1*	*nr*lr*
6	**do**speed[pezzo[i,1]]←0.	*1*	*nr*lr*
7	vettore[pezzo[i,1]]←mezzo	*1*	*nr*lr*
8	**return** vettore		

Questo è lo pseudocodice della prima euristica.

Il costo dell'istruzione numero quattro è $4lr+4$ perché al costo necessario per il calcolo dei quattro parametri utilizzati (velocità media, massima accelerazione e scarto quadratico medio), pari a $4lr$,va aggiunto il costo impiegato dall'albero di decisione per arrivare alle foglie, cioè l'altezza dell'albero stesso, che è *4*.Basandosi sulleconsiderazioni fatte il tempo T(*L*) di esecuzione del primo algoritmo è

$T(L) = nr (1 + L + 8lr + 4) \rightarrow \Theta(nr(L+ lr))$.

Questo vale a dire che il primo algoritmo classifica un tracciato di lunghezza *L* con un tempo di esecuzione lineare. Inoltre si può notare che il tempo di esecuzione per questo algoritmo non dipende dalla lunghezza del tracciato ma dal numero di intervalli che la funzione RangeV calcola a partire da questo.

	E2	**costo**	**numero di volte**
1	**While** j < Length[c]	*1*	*nr*
2	**Do** range←RangeV[speed,c[j]]	*lr*	*nr*
3	pattern←speed in range	*1*	*nr*
4	cartella←Scelta[pattern]	*3*	*nr*
5	**While** x < Length[cartella]	*1*	*5*nr*
6	**Do** files←LeggiCartellaDizionario[cartella[x]]	*nf*lf*	*5*nr*
7	**While** z < Length[files]	*1*	*5*nf*nr*
8	**Do** ts←LeggiVoceDizionario[files[z],cartella[x]]	*lf*	*5*nf*nr*
9	testo←ElaboraSegnale[ts]	*lf*	*5*nf*nr*
10	**While** (b+offset)<Length[pattern]	*1*	*5*nf*nr*(lr-b+offset)*
11	**Do** patt←Table[pattern da b a (b+offset)]	*offset*	*5*nf*nr*(lr-b+offset)*
12	n←n + kmpSearch[patt,testo]	*lf-lr*	*5*nf*nr*(lr-b+offset)*
13	b←b+ (b+offset)	*1*	*5*nf*nr*(lr-b+offset)*
14	z++	*1*	*5*nf*nr*
15	dir←Append[dir,n,cartella[x]]	*1*	*5*nr*
16	n←0	*1*	*5*nr*
17	x++	*1*	*5*nr*
18	mezzo←Append[mezzo,Max[dir]]	*1*	*nr*
19	**For** i←range[1] to range[2]	*1*	*nr*lr*
20	**Do** vettore[i]←mezzo	*1*	*nr*lr*
21	dir←{}	*1*	*nr*
22	j++	*1*	*nr*
23	**return** vettore		

Guardando lo pseudocodice del secondo algoritmo si vede già che la complessità rispetto al primo è maggiore. La parte che richiede maggiortempo è quella centrale dove l'algoritmo deve spezzare ognirange, che calcola, in pattern di lunghezza offset = 5. Per ognuno di questi pattern esegue la ricerca nel file, terminati i pattern cambia file e ricomincia.La funzione *LeggiCartellaDizionario*ha un costo pari a *nf*lf* perché deve leggere tuttigli *nf* file di lunghezza *lf*.Si nota che anche in questo caso il costo computazionale non dipende dalla lunghezza del tracciato in input ma dal numero di range calcolati.

Il tempo di esecuzione T(*L*) per il secondo algoritmo risulta

T(*L*) = nr(5+lr) + 5nr(1+nf*lf) + 5nr*f*n(2lf+5) + (5nr*f*n(lr-b+offset))* *(2+offset+lf-lr) + 2nr*lr

Escludendo le costanti e considerando l'offset, che rimane fissato a cinque, come tale questo costo può essere semplificato:

→nr*lr + nr(nf*lf) + nr*lf + nf*nr(lr)(lf-lr) + nr*lr

raccogliendo nr,

→nr (lr + (nf*lf) + lf + nf*lr(lf-lr) + lr)

→Θ(nr(lr + (nf*lf) + lf + nf*lr(lf-lr)))

In questo caso si vede come il tempo di esecuzione sia fortemente condizionato dal dizionario, cioè dal numero dei file presenti in esso e dalla lunghezza di questi, oltre che dal numero e dalla lunghezza dei range.

Il terzo algoritmo è il più complesso ma cerca di separare il lavoro che va svolto sul dizionario da quello per la classificazione.

Analizzeremo entrambe le fasi.

	E3 - Apprendimento	**costo**	**numero di volte**
1	**For** i←1 **to** Length[cartella]	*1*	*5*
2	**Do** files←LeggiCartellaDizionario[cartella[i]]	*nf*	*5*
3	**For** j←1 **to** Length[files]	*1*	*5*nf*
4	**Do** ts←LeggiVoceDizionario[files[j],cartella[i]]	*lf*	*5*nf*
5	testo←ElaboraSegnale[ts]	*lf*	*5*nf*
6	**While** (p+offset)<Lenght[testo]	*1*	*5*nf*(lf-p+offset)*
7	**Do** pattern←Table[testo da p a (p+offset)]	*offset*	*5*nf*(lf-p+offset)*
8	n←0.	*1*	*5*nf*(lf-p+offset)*
9	**For** z←1 **to** Length[files]	*1*	*5*nf*(lf-p+offset)*lf*
10	**Do** t←LeggiVoceDizionario[files[z],cartella[i]]	*lf*	*5*nf*(lf-p+offset)*lf*
11	t2←ElaboraSegnale[t]	*lf*	*5*nf*(lf-p+offset)*lf*
12	n←n+kmpSearch[pattern,t2]	*lf-offset*	*5*nf*(lf-p+offset)*lf*
13	{clas,lista}←Score[n,pattern]	*lc*	*5*nf*(lf-p+offset)*
14	v[i]←Table[class da 1 a Length[class]]	*lc*	*5*nf*
15	class←{}	*1*	*5*
16	lista←{}	*1*	*5*
17	SetParametri[v[1],v[2],v[3],v[4],v[5]]	*5*lc*	*1*
18	return		

Il costo per la fase di apprendmento è

T(n) = 5(3+nf) + 5nf(1+2lf+lc) + (5nf(lf-p+offset)(2+offset+lc)) + (5nf*lf(lf-p+offset)(1+ (lf-offset)+2lf)) + 5lc

→nf+nf(lf+lc) + nf*lf*lc + nf(lf*lf*(lf+lf))+lc

→ $\Theta(nf(1+ lf+lc+lf*lc+2lf^3)+lc)$, con lunghezza classifica uguale ad *lc*

La fase di apprendimento ha un costo computazionale cubico rispetto alla lunghezza dei file. Per costruire le classifiche dei pattern caratteristici la funzione deve leggere tutti i file del dizionario e per ogni file, calcolare tutti i pattern, di lunghezza prefissata, possibili. A questo punto, per ogni pattern che calcola richiama la funzione di matching (kmpSesrch) su tutti i file del dizionario per contare le occorrenza di quel pattern.

La funzione Score serve per la gestione delle classifiche temporanee per i pattern calcolati. La lunghezza della classifica è fissata all'inizio, per gli studi è stata usata una lunghezza (*lc*) pari a 100.

	E3 - Classificazione	**costo**	**numero di volte**
1	**Per ogni** range	*1*	*nr*
2	**Do** range←RangeV[speed,index]	*lr*	*nr*
3	vel←speed in range	*lr*	*nr*
4	**For** i←1 to 5	*1*	*5*nr*
5	**DoFor** j←1 to Length[pattern[i]]	*1*	*5*nr*lp*
6	**Do** cont[i]←cont[i]+kmpSearch[pattern[i,j],vel]	*L-lp*	*5*nr*lp*
7	app←Max[cont]	*1*	*nr*
8	**If** app = 0.	*1*	*nr/2*
9	**Do For** w←range[1] to range[2]	*lr*	*nr/2*
10	**Do** vettore[w]←"na"	*lr*	*nr/2*
11	**Else**	*1*	*nr/2*
12	**Do** pos←Position[contatore,app]	*1*	*nr/2*
13	**For** w←range[1] to range[2]	*lr*	*nr/2*
14	**Do** vettore[w]=pos	*lr*	*nr/2*
15	Contatore←{0,0,0,0,0}	*1*	*nr*
16	Return vettore		

Il costo per la fase di classificazione del terzo algoritmo è

$T(L) = nr(3+2lr) + 5nr + 5nr*lp(L-lp) + nr/2(4lr + 4)$

→$nr*lr + nr + nr*lp(l-lp) + nr*lr$

→$\Theta(nr*lr + nr + nr*lp(l-lp))$, con lunghezza pattern uguale a *lp*.

Questo algoritmo ha il pregio di demandare il carico di lavoro più pesante alla fase di apprendimento, che sì risulta molto complessa e lenta ma viene eseguita una sola volta. Al contrario, la seconda euristica applica questo carico sulla classificazione rendendola più lenta e laboriosa.

4. Risultati

In questa sezione sono mostrati i risultati del lavoro svolto.

Il problema fondamentale,comune a tutte le euristiche è la difficoltà nell'etichettare in maniera appropriata tracciati rilevati su mezzi diversi ma somiglianti tra loro.Questo accade quasi esclusivamente con tracciati catturati in autobus, bici e macchina.Soprattutto nei centri cittadini, fattori come il traffico e le code (causati da semafori, rotatorie, attraversamenti pedonali, ecc.) provocano queste situazioni ambigue. Magari un'auto, non avendo possibilità di sorpasso, è costretta a marciare dietro un autobus, a fermarsi alle soste con lui e ripartire con la stessa accelerazione o magari, un autobus rallentato dal traffico potrebbe mantenere la stessa andatura e addirittura avere le stesse accelerazioni di una bicicletta.

Anche la tecnologia svolge un ruolo importante: in questo momento i dispositivi utilizzati per la cattura dei percorsi sono già obsoleti e nuove versioni dei sistemi (di geolocalizzazione,per esempio) di cui questi dispongono sono stateaggiornate.

Per la costruzione delle tabelle dei risultati sono stati considerati dodici percorsi che rappresentano, in linea di massima, le caratteristiche dell'intero database.

I due valori nelle celle rappresentano le percentuali di etichettamento esatto calcolate rispettivamente con e senza la presenza di zeri (nel secondo caso si sono trascurati gli intervalli inattivi).

Tutti i risultati sono stati registrati senza l'utilizzo della funzione RiempiZeri, il motivo è che questa, in generale, aumenta la prima percentuale calcolata (quella che contempla gli intervalli d'inattività) mentre la seconda rimane invariata essendo calcolata trascurando gli zeri.Questi risultati sono ottenuti a partire dalle euristiche e servono per dare un'idea generale sull'efficienza di questi algoritmi; per ottenere dei dati realmente affidabili si dovrebbero prima implementare queste soluzioni utilizzando un linguaggio di

programmazione più adatto (come ad esempio il C) e poi condurre i test con una base di dati più ampia.

Tipologia tracciato	E1		
	rate=1	rate=2	rate=3
autobus+piedi	**78-92**	**80-93**	**82-93**
Auto1	**46-60**	**0-0**	**0-0**
Auto2	**56-62**	**56-62**	**56-62**
auto+piedi	**81-100**	**13-15**	**14-16**
Bici1	**100-100**	**100-100**	**100-100**
Bici2	**61-75**	**50-58**	**85-98**
Bus	**58-68**	**27-30**	**23-25**
bus+piedi	**35-42**	**10-10**	**20-19**
Piedi1	**97-100**	**97-100**	**98-100**
Piedi2	**97-100**	**100-100**	**100-100**
Treno1	**91-100**	**93-100**	**93-100**
Treno2	**89-94**	**89-94**	**90-94**

Tabella 1 – Risultati E1

Questa tabella (Tabella 1) presenta i risultati della prima euristica (E1).

L'euristica I è molto veloce ma come si vede dalla tabella, sono abbastanza frequenti casi in cui l'etichettamento è completamente (o quasi completamente) sbagliato.

Anche il rate gioca un ruolo importante; ci sono casi in cui, aumentando il rate, la soluzione migliora mentre in molti altri casi no. Capire con certezza quale sia il più appropriato comporterebbe uno studio più approfondito.

Tipologia tracciato	E2		
	rate=1	rate=2	rate=3
autobus+piedi	**92-93**	**84-94**	**83-93**
Auto1	**79-86**	**100-100**	**100-100**
Auto2	**94-100**	**83-89**	**98-100**
auto+piedi	**16-15**	**16-15**	**16-16**
Bici1	**100-100**	**100-100**	**100-100**
Bici2	**90-98**	**94-98**	**47-50**
Bus	**96-98**	**99-99**	**99-99**
bus+piedi	**55-61**	**25-29**	**0-0**
Piedi1	**100-100**	**100-100**	**100-100**
Piedi2	**100-100**	**100-100**	**100-100**
Treno1	**95-100**	**99-100**	**55-60**
Treno2	**92-94**	**100-100**	**100-100**

Tabella 2 – Risultati E2

In Tabella 2 sono riportati i risultati della seconda euristica, E2.

Questa è più efficace ma è anche la più lenta delle tre studiate.

L'etichettamento in questo caso è più raffinato e stabile anche se sono ancora presenti casi in cui il mezzo viene confuso.

Qui è ancora più evidente il fatto che si sono dei tracciati troppo simili tra loro; il problema della prima euristica è che si valutano dei parametri molto generici come la velocità media e lo scarto quadratico medio, in questo caso viene confrontato direttamente il segnale.

Un altro parametro oggetto di studi futuri è la lunghezza dei pattern utilizzati; come per il rate si nota una differenza nei risultati al variare di queste lunghezze. Per questa tabella è stata utilizzata una lunghezza del pattern pari a cinque.

Tipologia tracciato	E2_dist		
	rate=1	rate=2	rate=3
autobus+piedi	**44-45**	**43-44**	**40-47**
Auto1	**90-100**	**96-100**	**100-100**
Auto2	**10-10**	**11-10**	**8-6**
auto+piedi	**87-100**	**91-100**	**95-100**
Bici1	**100-100**	**100-100**	**100-100**
Bici2	**89-100**	**94-100**	**99-100**
Bus	**93-100**	**97-100**	**100-100**
bus+piedi	**58-64**	**65-68**	**76-80**
Piedi1	**98-100**	**99-100**	**100-100**
Piedi2	**98-99**	**100-100**	**100-100**
Treno1	**93-100**	**95-100**	**97-100**
Treno2	**98-100**	**100-100**	**100-100**

Tabella 3 – Risultati E2_dist

Questa è la tabella (Tabella 3) che riguarda i risultati della modifica fatta a E2.

Si è applicata una nuova funzione per il matching (vedi Paragrafo 3.2.2) per vedere se i risultati miglioravano. Teoricamente questa funzione è meglio formulata di quella utilizzata ma dando uno sguardo alle tabelle si nota che in realtà l'etichettamento è addirittura peggiorato. Questa funzione, probabilmente, sarebbe migliore di quella effettivamente usata se nel dizionario esistessero dei pezzi di tracciati molto simili ai pattern ricercati; gli etichettamenti errati scaturiscono dal fatto che la funzione sceglie i pattern con un valore di distanza che sia minore possibile ma questo non esclude il fatto che comunque sia tollerato un valore di soglia troppo alto. Con un valore di soglia (distanza tra i due tracciati) minore di 0.5 si cominciano ad avere risultati abbastanza tollerabili.

Tipologia tracciato	E3		
	rate=1	rate=2	rate=3
autobus+piedi	**75-86**	**84-89**	**83-89**
Auto1	**78-86**	**100-100**	**100-100**
Auto2	**81-89**	**82-89**	**98-100**
auto+piedi	**87-100**	**16-15**	**16-15**
Bici1	**100-100**	**100-100**	**100-100**
Bici2	**62-75**	**95-98**	**100-100**
Bus	**82-89**	**99-99**	**99-99**
bus+piedi	**55-60**	**25-29**	**25-29**
Piedi1	**100-100**	**100-100**	**100-100**
Piedi2	**100-100**	**100-100**	**100-100**
Treno1	**94-99**	**97-99**	**55-60**
Treno2	**92-94**	**100-100**	**100-100**

Tabella 4 – Risultati E3

Infine mostriamo la terza euristica, E3 (Tabella 4). I risultati mostrati sono stati ottenuti con lunghezza dei pattern pari a 2; con lunghezze superiori a 3 i pattern scelti dalla funzione di apprendimento non riescono a classificare in maniera appropriata i percorsi, questo perché i tracciati salvati nel dizionario non sono sufficienti.

Detto questo, analizzando la tabella si nota un leggero miglioramento sui tracciati più problematici ad E2 tuttavia alcuni tracciati hanno una classificazione leggermente peggiore. In linea generale questa euristica si adatta meglio ma è leggermente meno accurata in certe situazioni rispetto alla seconda.

Se escludiamo la fase di apprendimento, che sì è molto lenta, ma va eseguita sostanzialmente solo la prima volta, questa euristica ha la qualità di essere efficace quanto la

seconda ma ha un tempo di classificazione nettamente inferiore, quasi paragonabile alla prima.

In Tabella 5 sono riportati i tempi delle tre euristiche relativi al rate utilizzato, ricordiamo che il rate è l'intervallo di campionamento vale a dire ogni quanti secondi viene fatta una rilevazione.

Come per i risultati anche i tempi andrebbero rilevati dopo una reale implementazione del progetto.Il tempo è visualizzato in secondi arrotondato all'unità, è calcolato con un processore dual core (2.40 GHz) e serve a dare una stima dei tempi di elaborazione.

Rate	E1	E2	E3	
			Classificazione	*Apprendimento*
1	**43**	**410**	**75**	**1417**
2	**42**	**128**	**59**	**426**
3	**41**	**77**	**54**	**217**

Tabella 6 – Tempi di elaborazione

Questa tabella evidenzia come la prima euristica, sebbene poco accurata, sia la più veloce.

In realtà il commento che va fatto su questa tabella riguarda le altre due euristiche. Sebbene efficaci allo stesso modo, il tempo di classificazione di E3 rispetto a E2 è molto minore; questo fa della terza euristica l'approccio migliore.

Conclusioni e sviluppi futuri

L'obiettivo del lavoro svolto è quello di sviluppare algoritmi che favoriscano l'accesso a nuovi implementazioni. In questo caso il problema richiedeva di classificare i mezzi utilizzatidurante la percorrenza di un tracciato e quindi apparteneva ad un contesto ben specifico ma più in generale queste euristiche riguardano la classificazione per qualsiasi tipo di segnale discreto, si vuol dare quindi l'incipit in questo più ampio senso.

I risultati, discussi in precedenza, mostrano un buon comportamento degli algoritmi che potrebbero essere implementati senza fare ulteriori considerazioni.

Sono risultati abbastanza generali, ottenuti con un dizionario di trenta tracciati su un totale di cento circa, ma comunque abbastanza realistici.

Un aspetto fondamentale di questo studio riguarda appunto gli sviluppi che possono essere condotti a partire dai concetti discussi.

La prima euristica di per sé non ha grandi margini di miglioramento, ma data la velocità di elaborazione può essere utilizzata come supporto da un sistema più "intelligente" capace di prendere delle decisioni più accurate; per esempio, una volta visionato il risultato che E1 fornisce, se questo non è soddisfacente, si possono variare i parametri utilizzati ed "adattarli" alla nuova situazione di cui si è venuti a conoscenza o magari può essere utilizzata per avere "un' impressione" di partenza per poi eseguire altri calcoli o addirittura può essere utilizzata dalle altre due euristiche che ne faranno uso in quei casi dove queste non ottengono un buon risultato.

Il discorso vale anche per le altre due euristiche (E2 ed E3) che potrebbero essere messe nelle condizioni di capire quando e come elaborare il segnale in maniera ottimale o variare la lunghezza dei pattern secondo il tipo di tracciato che stanno analizzando. Data la presenza di un dizionario, si possono fare delle considerazioni aggiuntive; si potrebbero sviluppare delle caratteristiche che permettano una gestione intelligente di questo. Grazie a

funzioni apposite, il sistema potrebbe accorgersi che qualche voce all'interno del dizionario disturba la classificazione e quindi eliminarla o viceversa se è completamente sicuro che un etichettamento eseguito sia esatto potrebbe aggiungerlo ampliando così la sua esperienza. Azioni di questo tipo aiuterebbero a mantenere un dizionario efficiente e accurato che permetterebbe di sicuro un'altrettanto efficiente classificazione.

In generale si potrebbe dotare queste euristiche di una sorta di coscienza che dia la possibilità di decidere di cambiare il proprio comportamento a fronte di una nuova situazione creando così nuova esperienza.

Appendice

In questa sezione è riportato tutto il codice scritto in Mathematica utilizzato per la realizzazione delle euristiche.

Per facilitare la lettura, è riportato lo scheletro di una funzione che ne spiega la sintassi.

```
NomeFunzione [parametro1_,parametro2_,…]:=

 Module[{var1,var2,…},
  (
     …
   Codice
     …
  )
 ];
```

Figura A.1 - Sintassi di una funzione in Mathematica

```
E1[tracciato_]:=

 Module[{i,max,speed,pezzo,vettore,mezzo,tab},
  (

   tab=Flatten[CreaTabellaTracciato[tracciato,{"speed"}]];
   speed=Flatten[Table[tab[[i]],{i,1,Length[tab],rate}]];
   vettore=speed;
   While[Max[speed]>0.2,
    max=MaxV[speed];
    pezzo=RangeV[speed,max[[2]]];
    mezzo=Mezzo[pezzo];
```

```
      For[i=1,i<= Length[pezzo],i++,
       speed[[pezzo[[i,1]]]]=0.;
       vettore[[pezzo[[i,1]]]]=mezzo;
       ];
      ];
     Return[vettore]
     )
    ]
```

E2[tracciato_]:=

```
Module[{speed,range,vel,termine,patt,interv,pattern,p,attempt,P
max,files,offset,c,c1,i,n,v,x,z,j,r,b,q,g,cartella,testo,ts,dir
,mezzo,vettore,max,pos,s,tab},
   (

    (* inizializzazione *)
    x=1;
    interv=10;(*secondi in cui vel=0 *)
    tab=Flatten[CreaTabellaTracciato[tracciato,{"speed"}]];
    vel=Flatten[Table[tab[[i]],{i,1,Length[tab],rate}]];
    speed=vel;
    (* parametri *)
    offset=2;(* semi-ampiezza pattern*)
    termine=10;
    (*elaborazione segnale*)
    speed=ElaboraSegnale[speed,termine];
    vettore=Table["na",{i,1,Length[vel]}];
    pattern=speed;
    c=Punti[speed,offset];
    dir={};
    attempt=0;
    Pmax=0;(*punteggio max*)
    (*Print["punti da esaminare->",Length[c],"\n"];*)

    v=0;
    j=1;
    range=0;

    (*  ricerca  *)
    While[j<=Length[c],
     range=RangeV[speed,c[[j]],interv];
     If[vettore[[range[[1,1]]+interv]]==="na",
      mezzo={};
      pattern=Table[range[[i,2]],{i,1,Length[range]}];
      r=range;
```

```
      range={N[r[[1,1]]],N[r[[Length[r],1]]]};(*intervallo del
pattern in speed*)
      cartella=Scelta[pattern];
      x=1;
      n=0;
      While[x<=Length[cartella],(* ciclo di ricerca sulle
cartelle *)
       files=LeggiCartellaDizionario[cartella[[x]]];
       z=1;
       While[z<=Length[files],(*ciclo di ricerca sui file*)
        ts=LeggiVoceDizionario[files[[z]],cartella[[x]]];
        testo=Table[ts[[q]],{q,1,Length[ts],rate}];
        testo=ElaboraSegnale[testo,termine];
        b=1;
        While[(b+(2*offset))<=Length[pattern],
         patt=Table[pattern[[g]],{g,b,b+(2*offset)}];
         n+=kmpSearch[patt,testo];
         b+=(2*offset)+1;
         ];
        z++;
        ];
        dir=Append[dir,{n/Length[files],cartella[[x]]}];
       n=0;
       x++;
       ];
      (*- scrittura del vettore in output - *)
      max=Max[Table[dir[[i,1]],{i,1,Length[dir]}]];
      (*Print["max...",max];*)
      If[max>0,

pos=Flatten[Position[Table[dir[[i,1]],{i,1,Length[dir]}],max]];
       s=1;
       (*Print["pos->",pos];*)

For[s=1,s<=Length[pos],s++,mezzo=Append[mezzo,dir[[pos[[s]],2]]
]];
       (*Print["mezzo->",mezzo];*)
       ];

      For[i=range[[1]],i<range[[2]],i++,vettore[[i]]=mezzo;];
      ];
     dir={};
     j++;
     ];
    For[i=Length[vel]-
termine,i<=Length[vel],i++,vettore[[i]]=mezzo;];
    vettore=vettore//."Piedi"->"P";
    vettore=vettore//."Auto"->"M";
```

```
    vettore=vettore//."Treno"->"T";
    vettore=vettore//."Bus"->"A";
    vettore=vettore//."Bici"->"B";
    vettore=vettore//.ToExpression["{}"]->"na";
    Return[Flatten[vettore]];
    )
   ];
```

```
E3[tracciato_]:=
  Module[{pattern,vettore,speed,sp,index,i,j,w,contatore,v,
vel,pos,range,ret,interv,position,app},
   (
    interv=10;
    ret={"P","B","A","M","T"};
    contatore={0,0,0,0,0};

    (*carico patterns*)
    pattern=GetParametri[];
    sp=Flatten[CreaTabellaTracciato[tracciato,{"speed"}]];
    v=Table[sp[[i]],{i,1,Length[sp],rate}];
    speed=v;
    speed=ElaboraSegnale[speed];
    vettore=Table[N[0],{i,1,Length[speed]}];
    position=Flatten[Position[vettore,N[0]]];
    While[Length[position]>0,
     index=position[[1]];
     range=RangeV[speed,index,interv];
     vel=Table[range[[q,2]],{q,1,Length[range]}];
     For[i=1,i<=Length[pattern],i++,
      For[j=1,j<=Length[pattern[[i]]],j++,
        contatore[[i]]+=kmpSearch[pattern[[i,j]],vel];
        ];
      ];
     app=Max[contatore];
     If[app>N[0],
      For[w=range[[1,1]],w<=range[[Length[range],1]],w++,
        vettore[[w]]="na"];
      ,
      pos=Flatten[Position[contatore,app]][[1]];
      For[w=range[[1,1]],w<=range[[Length[range],1]],w++,
       vettore[[w]]=ret[[pos]];
       ];
      ];
     contatore={0,0,0,0,0};
     position=Flatten[Position[vettore,N[0]]];
     ];
    For[i=Length[speed]+1,i<=Length[v],i++,
```

```
      vettore=Append[vettore,vettore[[Length[speed]]]];];
     (*Print["len= ",Length[vettore]];*)
     Return[vettore];
     )
    ];
```

Apprendimento[]:=

```
Module[{dir,car,n,p,i,j,v,w,q,z,t1,punti,testo,lista,len,files,
classifica,offset,pattern,class,ts,t2},
    (
     classifica={};
     lista={};(*contiene i patter in classifica*)
     v={{},{},{},{},{}};
     class={};
     dir={"Piedi","Bici","Bus","Auto","Treno"};
     offset=5;
     len=50;(*numero pattern per ogni mezzo*)
     n=0;

     For[i=1,i<=Length[dir],i++,

car=SetDirectory["C:\\Users\\mattia\\Desktop\\tesi\\progetto
mob urb\\Dizionario\\"<>dir[[i]]<>"\\"];
      files=FileNames[];
      For[j=1,j<=Length[files],j++,
       ts=LeggiVoceDizionario[files[[j]],dir[[i]]];
       testo=Table[ts[[q]],{q,1,Length[ts],rate}];
       testo=ElaboraSegnale[testo];
       For[p=1,(p+offset-1)<=Length[testo],p++,
pattern=Table[testo[[w]],{w,p,p+offset-1}];
        n=0;
        For[z=1,z<=Length[files],z++,
         t2=LeggiVoceDizionario[files[[z]],dir[[i]]];
         t1=Table[t2[[q]],{q,1,Length[t2],rate}];
         t1=ElaboraSegnale[t1];
         n+=kmpSearch[pattern,t1];
         ];

{classifica,lista}=Score[n,pattern,classifica,lista,len];
        ];
       ];
      class=Table[classifica[[w,2]],{w,1,Length[classifica]}];
      v[[i]]=class;
      classifica={};
      lista={};
      ];
     SetParametri[v[[1]],v[[2]],v[[3]],v[[4]],v[[5]]];
```

```
    Return["Apprendimento eseguito"];
    )
   ];
```

```
Score[n_,pattern_,classifica_,occ_,len_]:=

  Module[{lista,esc,i,occs},
   (
    occs=occ;
    esc=True;
    lista=Sort[classifica,#1[[1]]> #2[[1]]&];
    If[Length[lista]<len,
     lista=Append[lista,{n,pattern}];
     If[!(MemberQ[occs,pattern]),
      occs=Append[occs,pattern];];
     esc=False;
     ,
     i=1;
     While[(esc && i<= Length[lista]),
      If[n>lista[[i,1]],
        If[(!(MemberQ[occs,pattern])),
         lista=Delete[lista,Length[lista]];
         lista=Append[lista,{n,pattern}];
         occs=Append[occs,pattern];
         ];
        esc=False;
        ,
        i++;
        ];
      ];
     ];
    Return[{lista,occs}];
    )
   ];
```

```
kmpSearch[parola_,testo_]:=

 Module[{m,t,i,p,lista,count,pos,cnd,T},

   (
   p=0;
   t=0;
```

```
  m=0;
  count=0;
  i=1;
  lista={};

  While[(m+i)<=Length[testo],
   t++;
   If[i>1,If[parola[[p+1]]testo[[m+i]],p++;];];
   If[parola[[i]]==testo[[m+i]],
    If[iLength[parola],
      count++;
      m=m+i;
      t=0;
      p=0;
      i=1;
      ,
      i++;
      ];
    ,
    If[p>0,
      m=m+i-p;
      i=(p+1);
      t=p;
      ,
      m=m+i;
      p=0;
      t=-1;
      i=1;
      ];
    ];
   ];
  Return[count]
  )
 ]
```

```
SetParametri[p_,b_,a_,m_,t_]:=
  Module[{dir,s},
   (

dir=SetDirectory["C:\\Users\\mattia\\Desktop\\tesi\\progetto
mob urb\\Dizionario\\"];
    s={p,b,a,m,t};
```

```
    Export["Patterns.json",s];
    Print["\n Parametri salvati!! \n "];
    )
   ];

GetParametri[]:=
  Module[{dir,s},
   (

dir=SetDirectory["C:\\Users\\mattia\\Desktop\\tesi\\progetto
mob urb\\Dizionario\\"];
    s=Import["Patterns.json","JSON"];
    Return[s];
    )
   ];

RiempiZeri[vettor_]:=

Module[{zeri,pos,neg,zero,mezzon,mezzop,negativo,positivo,i,sve
ttore,q,x,z},
   (
    svettore=vettor;
    q=1;
    x=1;
    z=Length[svettore];
    While[ svettore[[x]]□"na" ,x++;];
    While[svettore[[z]]□"na" ,z--;];
    For[i=x,i>0,i--,svettore[[i]]=svettore[[x+1]]];
    For[i=z,i<=Length[svettore],i++,svettore[[i]]=svettore[[z-
1]]];
    zeri=Flatten[Position [svettore,N[0]]];
    zeri=Flatten[Append[zeri,Flatten[Position
[svettore,"na"]]]];
    zeri=Sort[zeri];
    If[Length[zeri]>0,
     zero=zeri[[q]];
     ];

    While[q<=Length[zeri],
     mezzon="neg";
     mezzop="pos";
     i=0;
     zero=zeri[[q]];
     neg=zero;
     pos=zero;
     neg--;
     pos++;
```

```
      While[neg>0,
       If[svettore[[neg]]!=N[0]&& svettore[[neg]]!="na",
         mezzon=svettore[[neg]];
         negativo=neg;
         neg=-1;
         ,
         neg--
         ];
       ];
      While[pos<Length[svettore],
       If[svettore[[pos]]!=N[0] && svettore[[pos]]!="na",
         mezzop=svettore[[pos]];
         positivo=pos;
         pos=Length[svettore]
         ,
         pos++
         ];
       ];
      q++;
      If[mezzop===mezzon,
       For[i=negativo,i<=positivo,i++,svettore[[i]]=mezzop;];
       q--;
       q=q+(positivo-zeri[[q]]);
       ];
      ];
     Return[svettore]
     )
    ];
```

```
ElaboraSegnale[speed_]:=

  Module[{x},
   (
    x=MovingAverage[speed,10];
    x=Round[x,0.1];
    Return[x];
    )];
```

```
d[p1_, p2_] :=
  Module[{somma, i, q, r, ris, ris1},
   (
    somma = 0;
    r = 0;
    ris1 = 99;
    For[i = 1, (i + Length[p2] - 1) <= Length[p1], i++,
     somma = 0;
```

```
      For[q = 1, q <= Length[p2], q++,
       somma += Abs[p1[[q + i - 1]] - p2[[q]]]];
      ris = somma/Length[p2];
      If[ris < ris1,
       ris1 = ris;
       r = i;
       ]
      ];
    Return[{ris1, r}];
    )
   ];


CampoNumericoQ[x_]:=
  If[Length[Intersection[{x},CampiNumerici]]>0,Return[True],
   Return[False]
   ];

CampoNonNumericoQ[x_]:=
  If[Length[Intersection[{x},CampiNonNumerici]]>0,Return[True],
   Return[False]
   ];

TrasformaDataInSecondi[data_]:=

 Module[{sec,x,y,p,speed},
  (
   x=StringSplit[data," "][[2]];
   x=StringSplit[x,":"]//ToExpression;
   sec=x[[3]]+x[[2]] 60+x[[1]]3600;
   Return[sec];
   )
  ];


TrasformaDataInGiorni[data_]:=

 Module[{giorni,x,y,p,speed},
  (
   x=StringSplit[data," "][[1]];
   x=StringSplit[x,"-"]//ToExpression;
   giorni=x[[1]]+x[[2]] 31+x[[3]];
   Return[giorni];
   )
  ]

CreaTabellaTracciato[stracciati_,sequenzadicampi_]:=
```

```
Module[{tracciati,dopo,trovato,giorni,secondi,tracciatotrasform
ato,elementotrasformato,nuovitracciati,i,j,h,z,campoelemento,el
emento,campo,t,n,m},
  (
   tracciati=stracciati;
   t=tracciati;
   tracciatotrasformato={};
   m=Length[sequenzadicampi];
   For[h=1,h<=Length[t],h++,
    elemento=t[[h]];
    elementotrasformato={};
    For[j=1,j<=m,j++,
     campo=sequenzadicampi[[j]];
     trovato=False;
     For[z=1,z<=Length[elemento],z++,
      campoelemento=elemento[[z]];
      If[campoelemento[[1]]===campo,
       trovato=True;
       If[campo==="data",
        giorni=TrasformaDataInGiorni[campoelemento[[2]]];
secondi=TrasformaDataInSecondi[campoelemento[[2]]];
elementotrasformato=Append[elementotrasformato,{giorni,secondi}
],
        If[CampoNumericoQ[campo],
elementotrasformato=Append[elementotrasformato,
ToExpression[campoelemento[[2]]]];
         ];
        If[CampoNonNumericoQ[campo],
elementotrasformato=Append[elementotrasformato,campoelemento[[2
]]];
         ];];
       Goto[dopo];
       ];];
     Label[dopo];

If[trovato===False,elementotrasformato=Append[elementotrasforma
to,"na"]];];

tracciatotrasformato=Append[tracciatotrasformato,elementotrasfo
rmato]; ];
   Return[tracciatotrasformato];
   )
  ]
```

Plotta[tracciato_]:=

```
 Module[{z,speed,data,plo,short,long,f},
```

```
 (
  z=Flatten[CreaTabellaTracciato[tracciato,{"speed"}]];
  speed=Table[{i,z[[i]]},{i,1,Length[z]}];
  (*data=Table[z[[i,2,2]]-z[[1,2,2]],{i,1,Length[z]}];
  plo=Table[{data[[i]],speed[[i]]},{i,1,Length[speed]}];
  short=MovingAverage[plo,20];*)
  (*f=Interpolation[plo];*)
  ListLinePlot[speed,PlotLabel->"velocità",PlotRange->Full]
  )
 ]
```

```
Controllo[trac_,vet_]:=
 Module[{i,mezzo,cont,zeri,par,speed,vettore,tab,per,per0},
  (
   tab=Flatten[CreaTabellaTracciato[trac,{"mezzo"}]];
   mezzo=Flatten[Table[tab[[i]],{i,1,Length[tab],rate}]];
   tab=Flatten[CreaTabellaTracciato[trac,{"speed"}]];
   speed=Flatten[Table[tab[[i]],{i,1,Length[tab],rate}]];
   vettore=vet;
   vettore=vettore//."Autobus"->"A";
   vettore=vettore//."A piedi"->"P";
   vettore=vettore//."Piedi"->"P";
   vettore=vettore//."Auto"->"M";
   vettore=vettore//."Treno"->"T";
   vettore=vettore//."Bus"->"A";
   vettore=vettore//."Bici"->"B";
   mezzo=mezzo//."Autobus"->"A";
   mezzo=mezzo//."A piedi"->"P";
   mezzo=mezzo//."Piedi"->"P";
   mezzo=mezzo//."Auto"->"M";
   mezzo=mezzo//."Treno"->"T";
   mezzo=mezzo//."Bus"->"A";
   mezzo=mezzo//."Bici"->"B";

   cont=0;
   zeri=0;
   par=0;

   If[Length[mezzo]!=Length[vettore],
Print["LUNGHEZZE DIVERSE!!"]
,
Print["Lunghezza tracciati:  ",Length[vettore]]];
   For[i=1,i<=Length[mezzo],i++,
    If[vettore[[i]]==mezzo[[i]] ,
      cont++;
      If[speed[[i]]<0.1 ,
       par++;
```

```
      zeri++;
      ];
     ,
     If[speed[[i]]<0.1 ,zeri++;];
     ];
   ];

  per=N[(cont/Length[mezzo])*100];
  per0=N[((cont-par)/(Length[mezzo]-zeri))*100];
  (*Print["Hit: ",cont];
  Print["Zeri: ",zeri];
  Print["Percentuale:  ",per,"%"];
  Print["Percentuale -senza zeri- :  ",per0,"%"];
  Print[mezzo];*)
  Return[{per,per0}]
  )
 ]
```

```
MediaNozeri[speed_]:=

 Module[{i,vel},
  (
  i=2;
  vel=speed;
  While[i<Length[vel],
   If[vel[[i]]N[0],
    vel=Delete[vel,i];
    i=i-1;
    ];
   i++
   ];
  Return[Mean[vel]]
  )
 ]
```

```
SigmaNozeri[speed_]:=

 Module[{i,vel},
  (
  i=2;
  vel=speed;
  While[i<Length[vel],
   If[vel[[i]]===0.,
    vel=Delete[vel,i];
    i=i-1;
    ];
```

```
    i++
    ];
   Return[Sqrt[Variance[vel]]];
   )
  ]
```

```
Indagine[tracciato_]:=

  Module[{speed,sigma,Amax,Vmax,Vmedia},
   (
 speed=Flatten[CreaTabellaTracciato[tracciato,{"speed"}]];
    Vmedia=MediaNozeri[speed];
    Vmax=Max[speed];
    sigma=SigmaNozeri[speed];
    Amax=Acc[speed];
    Print["V media :  ",Vmedia];
    Print["V max   :  ",Vmax];
    Print["A max   :  ",Amax];
    Print["Sigma   :  ",sigma];
    )];
```

```
NomiFile[lista_,dir_]:=

 Module[{i,veralista,pos,par},
  (
   Print[lista];
   veralista={};
   For[i=1,i<=Length[lista],i++,
    If[StringTake[lista[[i]],-5]===".json",
      par=StringTake[lista[[i]],{StringLength[dir]+1,-6}];
      veralista=Append[veralista,par];
      ];
    ];
   Return[veralista]
   )
  ]
```

```
CreaVoceDizionario[tracciato_,dir_]:=

Module[{speed,file,w,n,Piedi,Bici,Auto,Bus,Treno,f,name,s,voci}
,
    (
     w=True;
     n=0;
     speed=Flatten[CreaTabellaTracciato[tracciato,{"speed"}]];
     Switch[dir,
      "Piedi",
Piedi=SetDirectory["C:\\Users\\mattia\\Desktop\\tesi\\progetto
mob urb\\Dizionario\\Piedi\\"];,
      "Bici",
Bici=SetDirectory["C:\\Users\\mattia\\Desktop\\tesi\\progetto
mob urb\\Dizionario\\Bici\\"];,
      "Auto",
Auto=SetDirectory["C:\\Users\\mattia\\Desktop\\tesi\\progetto
mob urb\\Dizionario\\Auto\\"];
      "Bus",
Bus=SetDirectory["C:\\Users\\mattia\\Desktop\\tesi\\progetto
mob urb\\Dizionario\\Bus\\"];,
      "Treno",
Treno=SetDirectory["C:\\Users\\mattia\\Desktop\\tesi\\progetto
mob urb\\Dizionario\\Treno\\"];
],
      w=False;
      ];
     If[w,
      file=NomiFile[FileNames[],dir];
      n=Length[file];
      If[n>0,
       f=ToExpression[Sort[file][[n]]]+1;
       ,
       f=1;
       ];
      name=dir<>ToString[f]<>".json";
      s=OpenWrite[name];
```

```
     Write[s,Flatten[speed]];
     Close[s];
     Return[s]
     ,
     Return["Wrong directory!"]
     ];
    )
   ];
```

```
RangeV[speed_,indice_,interv_]:=

  Module[{i,a,b,pausa,tab},
   (
    a=1;
    b=Length[speed];
    pausa=interv ;
    i=indice-1;
    While[i>1,
     While[speed[[i]]<=0.1&& i>2,
      pausa--;
      If[pausa< 1,a=i;Goto [dopo];];
      i--];
     pausa=interv;
     i--];
    a=i;
    Label[dopo];
    If[a-1<= interv,a=1;];

(*se rimane poco tracciato a sx, lo aggiungo*)

    i=indice+1;
    pausa=interv;
    While[i<Length[speed],
     While[speed[[i]]<=0.1 && i<Length[speed]-2,
      pausa--;
      If[pausa<1,b=i;Goto [fine];];
      i++];
     pausa=interv;
     i++];
    b=i;
    Label[fine];
    If[Length[speed]-b<=interv,b=Length[speed]];

(*se rimane poco tracciato a dx, lo aggiungo*)

    tab=Table[{x,speed[[x]]},{x,a,b}];
    Return [tab]
    ) ];
```

```
LeggiCartellaDizionario[cartella_]:=

  Module[{dir,vet},
   (
    vet={};
If[DirectoryQ["C:\\Users\\mattia\\Desktop\\tesi\\progetto mob
urb\\Dizionario\\"<>cartella<>"\\"]===True,

dir=SetDirectory["C:\\Users\\mattia\\Desktop\\tesi\\progetto
mob urb\\Dizionario\\"<>cartella<>"\\"];
     If[Length[FileNames[]]>0,
      vet=FileNames[];
      ];
     ];
    Return[vet];
    )
   ];


LeggiVoceDizionario[nome_,cartella_]:=

  Module[{dir,vet},
   (
    vet={};
    If[DirectoryQ["C:\\Users\\mattia\\Desktop\\tesi\\progetto
mob urb\\Dizionario\\"<>cartella<>"\\"]===True,

dir=SetDirectory["C:\\Users\\mattia\\Desktop\\tesi\\progetto
mob urb\\Dizionario\\"<>cartella<>"\\"];
     If[FileExistsQ[nome]===True,
      vet=Flatten[ReadList[nome]];
      ];
     ];
    Return [vet];
    )
   ];


Mezzo[pezzo_]:=

  Module[{maxV,media,sigma,speed,mezzo,acc},
   (
speed=Flatten[Table[pezzo[[i,2]],{i,1,Length[pezzo]}]];
    media=MediaNozeri[speed];
    maxV=Max[speed];
    sigma=SigmaNozeri[speed];
    acc=Acc[speed];
```

```
        mezzo="na";
        If[maxV<= 35,
         If[media<10,
           If[sigma<3,
             mezzo="P";
             ,
             If[acc<8,
               mezzo="B";
               ,
               mezzo="M";
               ];
             ];
           ,
           If[sigma<6,
             mezzo="B";
             ,
             If[acc<13,
               mezzo="A";
               ,
               mezzo="M";
               ];
             ];
           ];
         ,
         If[acc<7,
           If[media<85,
             If[sigma<18,
               mezzo="A";
               ,
               mezzo="M";
               ];
             ,mezzo="T";
             ];
           ,
           If[maxV<80,
             If[acc>9 && sigma>14,mezzo="M";
               ,
               mezzo="A";
               ];
             ,
             mezzo="M";
             ];
           ];
         ];
        Return[mezzo]
        )
    ];
```

```
MaxV[tab_]:=

 Module[{max,indice},
   (
    max=Max[tab];
    indice=Max[Position[tab,max]];
    Return [{max,indice}];
    )]

Acc[speed_]:=

  Module[{tab,med,a,x,n},
    (
     tab=Table[(speed[[x+1]]-
speed[[x]])/rate,{x,1,Length[speed]-1}];
     med=Max[tab];
     Return[med];
     )
    ];

Accs[speed_,offset_]:=

  Module[{tab,a,x,acc,i},
    (
     acc={};
     tab=Table[((speed[[x+1]]-
speed[[x]])/rate),{x,1,Length[speed]-1}];
     i=offset;
     x=offset;
     While[i<=Length[tab]-offset,
      If[tab[[i]]>0,
       x=i;
       While[tab[[i]]>0 && i<=Length[tab]-offset,i++;];
       acc=Append[acc,Floor[((x+i+1)/2)]];
       ];
      i++;
      ];
     For[i=1,i<Length[acc],i++,
      If[acc[[i]]+offset<acc[[i+1]]-offset,
        acc=Append[acc,(acc[[i]]+offset+
acc[[i+1]]-offset)];];
      ];
     If[acc[[1]]-offset>offset,acc=Append[acc,Floor[(acc[[1]]-
offset)/2]]];
     acc=Sort[acc];
     Return[acc];
     )  ];
```

```
Punti[speed_,offset_]:=

  Module[{i,c},
   (
    i=offset+2;
    c={};
    While[i<(Length[speed]-offset),
     If[(speed[[i]]-speed[[i-1]])>0,
      While[speed[[i]]-speed[[i-1]]>0&&
i<(Length[speed]-offset),i++;];
      c=Append[c,N[i-1]];
      i++;
      ];
     If[(speed[[i]]-speed[[i-1]])<0,
      While[speed[[i]]-speed[[i-1]]<0&&
i<(Length[speed]-offset),i++;];
      c=Append[c,N[i-1]];
      i++;
      ];
     If[(speed[[i]]-speed[[i-1]])==0,i++;];
     ];
    c=Sort[c];
    Return[c];
    )];


Scelta[speed_]:=

  Module[{sigma,maxv,dir},
   (
    sigma=SigmaNozeri[speed];
    maxv=Max[speed];
    dir={"Piedi","Bici","Auto","Treno","Bus"};
    If[(sigma>5&&maxv>50),
     dir={"Auto","Treno","Bus" }
     ,
     If[( sigma<5 && maxv<15),
       dir={"Piedi","Auto","Bus","Bici"};
       ,
       If[(3<sigma<6 && maxv<35),dir={"Bici","Auto","Bus"};
         ];
        ];
     ];
    (*Print["Scelte cartelle ..>",dir];*)
    Return [dir];
    ) ];
```

Bibliografia

[1] Te-Ming Huang, V. Kecman, I.Kopriva," *Kernel Based Algorithms for Mining Huge Data Sets: Supervised, Semi-supervised, and Unsupervised Learning*", Springer, 13/apr/2006

[2] C.Szepesvári, "*Algorithms for Reinforcement Learning*", Morgan & Claypool Publishers, 01/ott/2010

[3] E.R.Hruschka,R.J.B.Campello,A.A.Freitas,A.C.Carvalho," *A Survey of Evolutionary Algorithms for Clustering*",IEEE

[4] R. Xu,"*Survey of Clustering Algorithms*" , IEEE and Donald Wunsch II, Fellow, IEEE

[5] L.Cayton," *Algorithms for manifold learning*", lcayton@cs.ucsd.edu,June 15, 2005

[6] S. Thrun,"*Learning To Learn: Introduction*", www.cs.cmu.edu,1996

[7] L. de Raedt, P. Flach, "*Machine Learning: ECML 2001*", Springer, 02/ott/2001

[8] B. Yegnanarayana,"*Artificial Neural Networks*",PHI Learning Pvt. Ltd., 01/ago/2004

[9] R.Riolo, B.Worzel, "*Genetic Programming Theory and Practice*", Springer, 01/dic/2003

[10] N. Cristianini, J. Shawe-Taylor,"*An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*", Cambridge University Press, 23/mar/2000

[11] P.Arabie, L. J. Hubert, G. De Soete,"*Clustering and Classification*", World Scientific, 1996

[12] T. Koski, J. Noble,“*Bayesian Networks:An Introduction”,* John Wiley & Sons, 26/ago/2011

[13] T.H.Cormen,C.E.Leiserson,R.L.Rivest,C.Stein,”*Introductionto algorithms*”, Seconda Edizione, McGraw-Hill,2005

[14] Standard ECMA-262, Terza Edizione - Dicembre 1999.

[15] S.Wolfram, ”*Wath is Mathematica?*”, http://www.wolfram.com/mathematica/, 2012.

[16] F Baader, T. Nipkow*, “Term-rewriting and all that”*, Cambridge University Press, 1999.

[17] A.Apostolico, Zvi Galil,“*Pattern Matching Algorithms* ”, Oxford University Press, 1997.

[18] N.Bassiliades, G.Governatori, A.Paschke, *“Rule-Based Reasoning, Programming, and Applications*”, Springer, 28/set/2011.

Ringraziamenti

Il primo, più importante, doveroso e sentito ringraziamento va ai miei genitori che si sono spaccati la schiena per fornire a me un'adeguata istruzione, non so se riuscirò a ripagarli mai abbastanza.

Ringrazio anche Giorgia, che ha tenuto alto il mio morale, a volte anche solo con la sua presenza, senza di lei sarei impazzito, e ringrazio Niki, per tutte quelle volte che l'ho scomodato dalla sedia coi miei dubbi farneticanti.

Ringrazio Gallo, per il fondamentale supporto tecnico.

In generale ringrazio tutti gli amici che mi hanno sopportato negli ultimi mesi.

Infine, un ringraziamento speciale va a Luciano Margara che, più che un professore e relatore, è stato per me amico e mentore.