Alma Mater Studiorum · Università di Bologna

SCUOLA DI INGEGNERIA E ARCHITETTURA

Dipartimento di Informatica – Scienza e Ingegneria

CORSO DI LAUREA IN INGEGNERIA INFORMATICA

TESI DI LAUREA

in

SISTEMI INTELLIGENTI

CLUSTERING DI TRACCE DI MOBILITÀ PER L'IDENTIFICAZIONE DI STILI DI GUIDA

CANDIDATO
Alessandro Petraro

RELATORE:
Michela Milano

CORRELATORI:
Federico Caselli, Marco Lippi

Anno Accademico 2016/2017

Sessione III

Alla mia famiglia

che da sempre mi sostiene ...

Abstract

Traffic simulators are effective tools to support decisions in urban planning systems, to identify criticalities, to observe emerging behaviours in road networks and to configure road infrastructures, such as road side units and traffic lights. Clearly the more realistic the simulator the more precise the insight provided to decision makers. This paper provides a first step toward the design and calibration of traffic micro-simulator to produce realistic behaviour. The long term idea is to collect and analyse real traffic traces collecting vehicular information, to cluster them in groups representing similar driving behaviours and then to extract from these clusters relevant parameters to tune the microsimulator. In this paper we have run controlled experiments where traffic traces have been synthetized to obtain different driving styles, so that the effectiveness of the clustering algorithm could be checked on known labels. We describe the overall methodology and the results already achieved on the controlled experiment, showing the clusters obtained and reporting guidelines for future experiments.

Indice

Elenco delle figure

Elenco delle tabelle

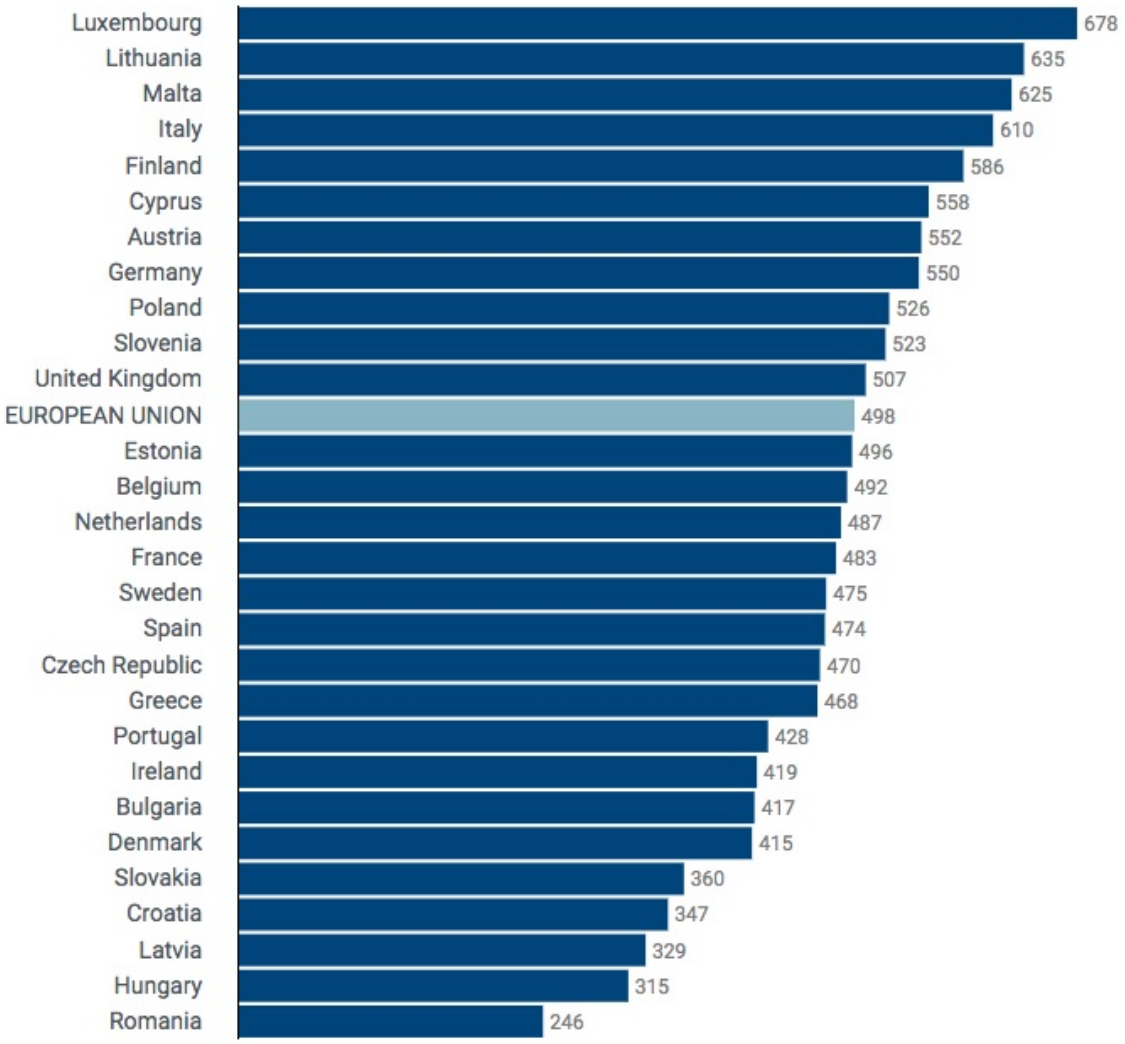
Introduzione

Negli ultimi anni l'analisi delle problematiche relative alla circolazione urbana delle autovetture ha assunto un ruolo via via sempre più importante. Secondo l'organizzazione americana Ward's il numero di veicoli motorizzati nel mondo è passato da 250 milioni di unità nel 1970 a 1.105 miliardi nel 2010, approssimativamente raddoppiando tra il 1950 e il 1970 per poi continuare a crescere in modo incrementale di circa 100/150 milioni di unità ogni 10 anni. Due ricercatori americani stimano che la flotta mondiale di veicoli raggiungerà i 2 miliardi entro il 2020 con le automobili a rappresentare il 50% di tutti i veicoli [1].

Secondo la European Automobile Manufacturers Association (ACEA), nel 2014 il numero di veicoli ogni 1000 abitanti in Italia si aggirava attorno alle 610 unità. Si aggiudicava invece il primato il Lussemburgo con 678 veicoli ogni 1000 abitanti seguito da Lituania (635) e Malta (625). La cosa interessante che emerge da questi dati è che la media europea si attestava a 498 unità indicando di come già nel 2014 una persona su due disponesse di un veicolo motorizzato [2].

È importante osservare di come questo possa avere chiaramente delle ripercussioni significative sul traffico stradale e sul numero di incidenti. Secondo l'organizzazione mondiale della sanità (OMS) gli incidenti stradali hanno causato 1.25 milioni di morti nel mondo nel 2013, ovvero circa 1 ogni 25 secondi, costituendo la principale causa di morte per persone tra i 15 e 29 anni e costando al governo approssimativamente un 3% del prodotto interno lordo. Sempre secondo l'OMS, per ogni persona che muore in un incidente stradale ve ne sono altre 20 che invece riportano ferite non mortali che possono però avere un impatto considerevole sulla loro qualità di vita e che spesso si traducono per lo stato in ulteriori costi economici [3]. Secondo dati ISTAT, in Italia nonostante il netto calo della mortalità rispetto al periodo gennaio-giugno 2015, il livello resta elevato e non in linea con quanto previsto dall'obiettivo europeo per il 2020 (dimezzamento del numero di vittime registrate nel 2010) [4].

Figura 1: Veicoli ogni 1000 abitanti - Dati 2014



L'aumento dei veicoli in circolazione ha chiaramente portato anche ad una maggiore diversificazione tra i possessori di un mezzo di un trasporto rendendo interessante lo studio di comportamenti emergenti e di stili di guida tra persone diverse. Alla base di questa tesi vi è infatti l'idea che poter identificare e in seguito comprendere e studiare questi comportamenti possa essere di ausilio alla realizzazione di modelli più attinenti alla realtà e di conseguenza avere effetti benefici non solo sulla problematica del traffico, ma in maniera indiretta, anche sulla sicurezza stradale e sulla pulizia dell'aria nelle nostre città. Si è dunque cercato di capire se, a partire da dati di interesse, fosse possibile estrarre dei comportamenti emergenti, ovvero identificare delle tipologie di guidatori. In altre parole, ci si è chiesti se fosse possibile o meno, a partire da informazioni come velocità e accelerazione per un veicolo, sia in forma

di serie temporale sia in forma aggregata, identificare dei cluster rappresentativi per una categoria di guidatori. A questo fine si è scelto di utilizzare un simulatore di traffico e in particolare la scelta è ricaduta su SUMO (Simulation Of Urban Mobility), un simulatore di traffico open-source sviluppato dal centro Aerospaziale Tedesco (DLR) con il quale si è simulato il movimento di un determinato numero di veicoli all'interno di diversi reticoli stradali reali. Sono stati poi realizzati degli strumenti che fossero in grado, a partire da questi output di simulazione, di determinare la presenza o meno di cluster rilevanti.

Nella presente trattazione, in primo luogo si fornirà una breve panoramica di quelle che sono le tipologie di modelli di traffico esistenti in letteratura. In seguito, l'attenzione verterà sui modelli microscopici, dei quali si delineeranno alcune implementazioni al fine di mostrare le motivazioni che hanno portato alla scelta di SUMO. Saranno poi trattati con maggiore dettaglio SUMO e le sue caratteristiche. In seguito, saranno descritte brevemente le principali tipologie di algoritmi di clustering allo scopo di evidenziare i motivi che hanno condotto alla scelta di k-means. Infine ci si concentrerà sulle simulazioni effettuate e sui risultati ottenuti, dando particolare rilievo alle problematiche incontrate nel flusso di lavoro e di come sono state risolte. Per chi fosse interessato, è inoltre reso disponibile in appendice B qualche dettaglio circa il sistema implementato.

Capitolo 1

I modelli di traffico

1.1 Le tipologie dei modelli di traffico

Tra gli strumenti che si possono utilizzare al fine di analizzare e individuare i problemi relativi alla mobilità urbana un posto di rilievo è dato sicuramente ai programmi di simulazione. In particolare, un simulatore è uno strumento software che permette di modellare un sistema reale che per qualche motivo non possiamo o non vogliamo usare direttamente e che astrae al contempo dai dettagli superflui e si concentra sui parametri e sugli aspetti di interesse.

È noto di come la complessità dei flussi di traffico la renda difficile da modellare con gli approcci matematici tradizionali favorendo invece l'uso di simulatori di traffico. Gli elementi che compongono il sistema, infatti, oltre ad essere molteplici (pedoni, veicoli, sistemi di controllo semaforico...), interagiscono tra loro in diversi modi andando ad aggravare la complessità del sistema stesso.

I simulatori di traffico sono stati ampiamente studiati in letteratura e possono essere classificati in quattro tipologie [5]:

1. Modelli macroscopici –> I modelli di traffico macroscopici sono basati sulla relazione deterministica esistente fra portata, velocità e densità. In tale approccio si tende a rappresentare i flussi di traffico usando modelli matematici di alto livello spesso derivati dalla fluido dinamica. Questo tipo di simulazione gestisce ogni veicolo allo stesso modo e questo rappresenta la sua principale limitazione. Si ha infatti che questo approccio non si configura come molto realistico in quanto nel mondo reale, tipicamente, non

solo i veicoli sono molto eterogenei fra loro in termini di caratteristiche fisiche, ma sono anche guidati da individui che si distinguono per stile di guida e comportamento.

2. Modelli microscopici –> I modelli di traffico microscopici si concentrano sui movimenti dei singoli veicoli, andando a modellare le interazioni tra automobilisti e tra automobilisti e l'ambiente stradale. In un tale approccio, la scelta del percorso, le decisioni riguardanti accelerazione o cambio di corsia di ciascun veicolo sono, in generale, modellate esplicitamente. Inoltre, ciascuna entità del flusso ha le proprie caratteristiche che possono includere: le caratteristiche del veicolo, come ad esempio l'accelerazione e la velocità massima, e le caratteristiche del conducente, come ad esempio il tempo di reazione o il tasso di impazienza.

3. Modelli mesoscopici –> I modelli di traffico mesoscopici si collocano in un punto intermedio fra i modelli macro e micro appena visti. In particolare, in tali modelli l'unità per il flusso di traffico è il singolo veicolo, ma i veicoli possono anche essere raggruppati e visti come una sola entità.

4. Modelli nanoscopici–> I modelli di traffico nanoscopici si collocano un livello sotto i modelli microscopici andando a dividere ulteriormente i veicoli nelle loro parti. Tale approccio è tipicamente usato in stretta relazione con la robotica dove si sta cercando, sopratutto negli ultimi anni, di realizzare sistemi in grado di guidare in modo autonomo.

Per lo scopo di questo elaborato si è scelto di concentrarsi su simulatori microscopici in quanto era necessario l'elemento veicolo come entità a se stante ed era molto importante che essi potessero differenziarsi il più possibile tra loro sia per caratteristiche fisiche che per caratteristiche comportamentali.

1.2 Panoramica simulatori microscopici

Prima di procedere oltre andiamo a definire una breve panoramica, che chiaramente non può essere esaustiva, dei simulatori di traffico microscopici in modo da andare ad evidenziare come è stata operata la scelta di quale utilizzare:

1. VISSIM [6]: Sviluppato da PTV in Germania, questo simulatore offre flessibilità sotto diversi aspetti e in particolare offre la possibilità di definire degli attributi e delle caratteristiche in modo individuale per l'automobilista e il veicolo. Tale software non è però open-source e anzi, può essere usato a titolo gratuito solo in versione di prova a 30 giorni. In alternativa ne può essere chiesta una copia gratuita per attività di tesi previa richiesta via email

2. PARAMICS [7]: Sviluppato a partire da un progetto dell'università di Edimburgo, è stato realizzato per un ampio spettro di casistiche nelle quali la congestione è tra le caratteristiche predominanti. Tale software non è open-source distribuito in diverse versioni in genere a pagamento. In alternativa, offre delle licenze per l'università al fine di permettere agli studenti di usarlo all'interno del campus

3. AIMSUN [8]: Sviluppato e venduto da TSS, offre una grande granularità dando la possibilità di realizzare simulazioni a livello microscopico, macroscopico e mesoscopico. Tale software è distribuito in versione di prova a 30 giorni oppure a pagamento

4. MITSIM [9]: Sviluppato al MIT sotto licenza omonima, permette di modellare i componenti della rete (strade, sistemi di controllo semaforico...), la demand di traffico e diverse condotte di guida. In particolare, per ogni singolo veicolo è possibile definire alcuni parametri comportamentali come ad esempio l'aggressività. Tale software è open-source, ma può compilare ed eseguire solo su sistemi operativi linux. In particolare, è indicato che per compilare il codice sorgente è necessario usare la distribuzione Redhat Linux 7.3. [10]

5. "Simulation of Urban Mobility" (SUMO) [11]: Sviluppato dal centro aerospaziale tedesco, è un simulatore open-source rilasciato sotto licenza GPL. Tra le sue caratteristiche permette di eseguire simulazioni a livello microscopico e, a partire dalla versione 0.26.0 anche a livello mesoscopico [12]. Inclusi con SUMO vi sono poi una serie di strumenti di supporto che possono essere usati ad esempio per generare percorsi, veicoli e reticoli stradali. Infine vi sono anche una serie di API che permettono di interfacciarsi direttamente con il simulatore stesso rendendo possibile l'estrazione di dati personalizzati per ogni simulazione.

La scelta alla fine è ricaduta su SUMO in quanto era uno dei pochi software open-source e rilasciato sotto licenza GPL. Inoltre, SUMO offre un'interfaccia nota come TRACI (Traffic Control Interface) che permette di accedere ai dati della simulazione in tempo reale. Per concludere, SUMO era già stato utilizzato in contesto universitario per un progetto europeo e dunque la scelta è stata anche strategica in quanto il suo uso ha comportato una curva di apprendimento meno ripida.

1.3 I limiti dei simulatori esistenti

È importante osservare di come i comportamenti degli automobilisti possano variare in modo anche molto considerevole in funzione, non solo della locazione geografica che si prende in considerazione, ma anche, ad esempio, dell'età del soggetto alla guida o dalla sua esperienza. Quando si tratta di persone, infatti, entrano tipicamente in gioco anche fattori psicologici e comportamentali che, in quanto legati al singolo individuo, risultano molto complessi da modellare in modo rigoroso [13]. Si pensi, a titolo puramente esemplificativo, di come non tutte le persone abbiano la stessa sicurezza, la stessa prudenza o la stessa aggressività nella guida oppure di come diverse strade possano comportare diverse reazioni in coloro che guidano. Questo rappresenta evidentemente un grande limite per un simulatore, il quale difficilmente riuscirà a riprodurre dei comportamenti che possano essere realistici per ogni demand di traffico simulata.

Un altro dei problemi riconosciuti per i simulatori e che si può collocare in questo tema, è quello della congestione del modello [14]. I simulatori esistenti, tipicamente, usano modelli che non sono realistici quando si tratta di congestione. Si può pensare, infatti, di come la congestione determini comportamenti diversi nelle persone coinvolte. Laddove alcune staranno ferme in coda, ve ne saranno altre che cercheranno di inserirsi nel primo spazio disponibile.

È chiaro che i risultati di una simulazione possono dare solo un'indicazione di quello che sarà il comportamento del sistema, ma è ancora più evidente di come, potendo inserire degli stili di guida reali all'interno di un simulatore, questo potrebbe dare un'indicazione quasi sicuramente più corretta.

1.4 Verso simulatori realistici

L'idea alla base di questa tesi è quella di analizzare tracce estratte da una serie di veicoli e di eseguirne un clustering sulla base di parametri che potrebbero poi essere utilizzati in seguito all'interno di un simulatore per definire veicoli dal comportamento simile. In altre parole, se un simulatore offrisse la possibilità di definire l'accelerazione massima per un veicolo e si riuscisse ad identificare un certo numero di stili di guida sulla base proprio dell'accelerazione a partire da delle tracce reali, allora si potrebbe usare proprio quelle accelerazioni estratte per realizzare una domanda di traffico che fosse più realistica.

In questa tesi, si è deciso di utilizzare il simulatore SUMO al fine di realizzare lo step di generazione delle tracce in quanto l'uso di tracce sintetiche si è ritenuto più adatto per l'obiettivo di questo studio. Si ha infatti che, conoscendo a priori le tipologie di veicoli nella rete in quanto inserite manualmente, si è potuto realizzare un clustering cercando di verificare se emergessero o meno dei cluster sulla base delle tipologie dei veicoli.

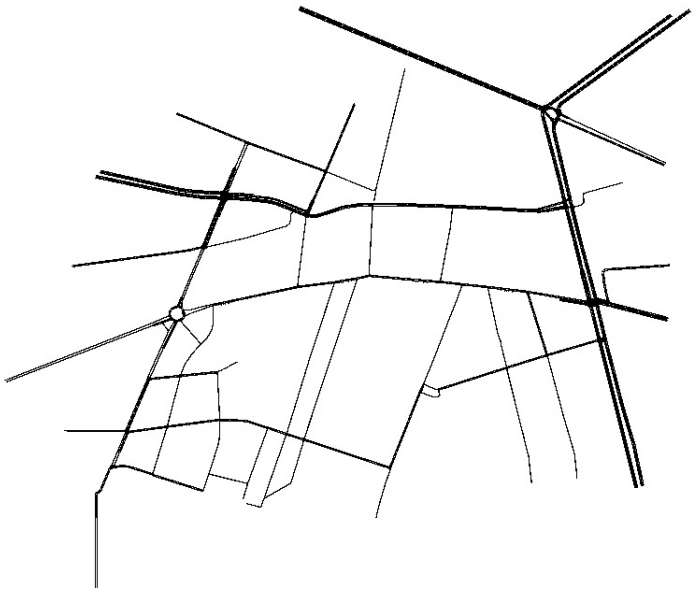
Capitolo 2

SUMO

2.1 La rete di traffico

In SUMO, una rete di traffico è definita dall'insieme dei nodi che la compongono, ognuno caratterizzato da un identificativo, una coordinata x, una coordinata y e un tipo.

Figura 2.1: Reticolo Andrea Costa



In particolare, il tipo specifica la gestione dell'incrocio stradale che ciascun nodo può rappresentare (nodo controllato da un semaforo, nodo con precedenza a destra...). I nodi sono poi collegati tra loro definendo quelle che saranno le effettive strade del reticolo. Tali collegamenti, in sumo sono chiamati edge e si compongono di un identificativo, un nodo di

partenza, un nodo di destinazione e un tipo che specifica la tipologia di strada identificata dall'edge stesso. Ad esempio potremmo avere autostrade, percorsi pedonali...

Per quel che riguarda il progetto, si è scelto di utilizzare due reticoli puramente sintetici, nel caso specifico una griglia 4X4 e un corridoio, e un reticolo reale, una parte di via Andrea Costa della città di Bologna. In particolare, mentre i reticoli sintetici sono stati usati solo al fine di acquisire conoscenza delle potenzialità di SUMO, il reticolo reale di Andrea Costa (Figura 2.1) è quello sul quale ci si è concentrati maggiormente ai fini della sperimentazione.

2.2 I veicoli

Dalla documentazione si legge che un veicolo in SUMO comprende tre parti [15]:

1. Il tipo che ne descrive le caratteristiche fisiche come ad esempio accelerazione, decelerazione, velocità massima... ecc. In particolare, esso può essere condiviso da più veicoli. Se non indicato, SUMO ne prevede uno generale di Default

2. La Route che il veicolo dovrebbe seguire come insieme di edge adiacenti. Anch'essa, come il tipo, può essere condivisa permettendo la generazione di veicoli con lo stesso percorso

3. Il veicolo stesso

Un veicolo SUMO può inoltre essere assegnato ad una classe astratta, che oltre a permettere una differenziazione dei veicoli considerati, può essere usata nella definizione delle corsie permettendo o meno il loro utilizzo da parte di determinati tipi di veicoli. Ad esempio, si potrebbe pensare di avere tre corsie delle quali quella di destra possa essere usata solo da taxi e autobus. Per quel che concerne questa tesi si è scelto di usare la classe di default "passenger" che denota un normale veicolo e che è stata considerata come la più rilevante ai fini di questo studio in quanto la più numerosa in strade reali.

In quanto appartenente alla tipologia dei microsimulatori, SUMO concede poi di raggiungere un livello di granularità molto fine circa la modellazione dei singoli veicoli che verranno simulati permettendo di definire delle classi di veicoli personalizzate ognuna con i propri tratti distintivi. Tali classi possono poi essere assegnate ad una probabilità di generazione che ne determinerà la presenza in termini di quantità all'interno della rete.

Una funzionalità che si è dimostrata di estrema rilevanza per questo progetto è stata la possibilità offerta da SUMO di definire non solo le caratteristiche fisiche per la tipologia di veicolo come ad esempio la lunghezza o la velocità massima raggiungibile, ma anche l'aggressività della classe definita. In particolare, si ha che in SUMO, la velocità massima che un veicolo può raggiungere in una determinata strada è calcolata come il prodotto tra un moltiplicatore definito per quel veicolo e la velocità massima consentita su quella specifica strada. Tale moltiplicatore, però, è ottenuto a sua volta da una distribuzione normale di probabilità dove media e deviazione standard possono essere configurate a partire da due attributi di classe noti rispettivamente come Speed Factor e Speed Deviation. Ad esempio, usando speedFactor=1.0 e speedDev=0.1 otterremo una distribuzione di velocità nella quale il 95% dei veicoli viaggerà tra l'80% e il 120% del limite legale [16] (Figura 2.2 e figura 2.3).

Figura 2.2: Distribuzione normale

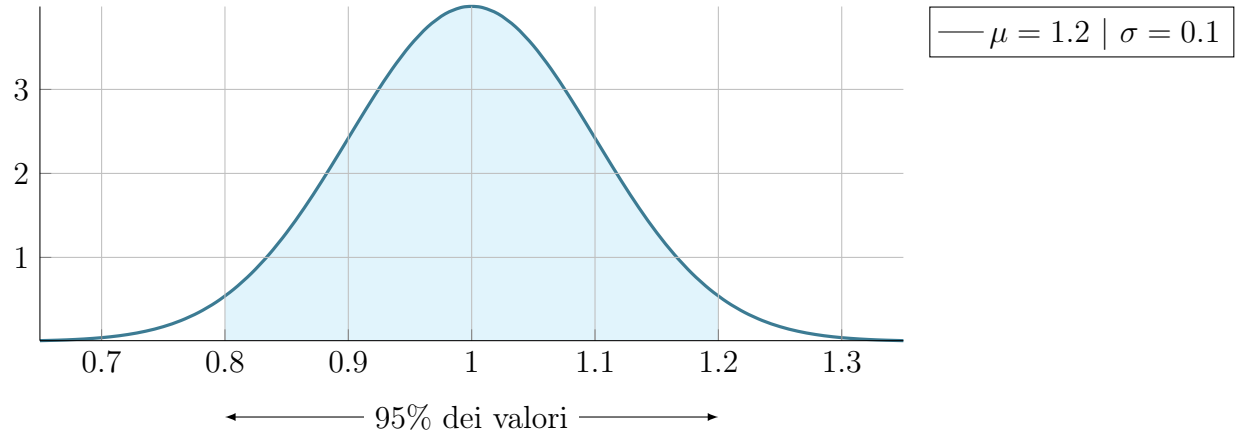
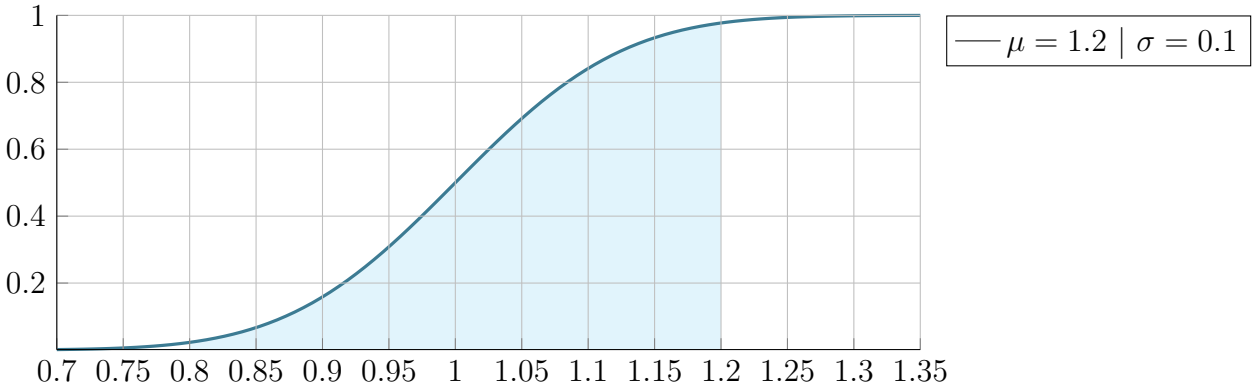


Figura 2.3: Funzione di distribuzione



Osserviamo di come l'utilizzo di Speed Factor elevati risulti in una maggiore probabilità che gli automobilisti appartenenti ad una determinata classe siano più aggressivi, e di come, viceversa, Speed Factor bassi portino ad avere automobilisti più rispettosi dei limiti di velocità. Tale attributo è stato dunque usato ampiamente nella tesi al fine di ottenere comportamenti realistici all'interno della rete.

Per quel che concerne la Speed Deviation, l'osservazione che possiamo fare è che mantenendo questo valore basso avremo classi di veicolo maggiormente omogenee, mentre alzandolo avremo automobilisti con una maggiore varietà di moltiplicatori di velocità anche all'interno della stessa classe. Ai fini di questo progetto, si è dunque optato per un valore che favorisse l'omogeneità intra-classe in modo da evitare che una stessa classe di veicoli avesse stili di guida differenti.

Per quel che riguarda i parametri discriminatori tra una tipologia di veicolo e l'altra, la scelta è ricaduta su quelli che sarebbero più facilmente reperibili anche in una situazione reale. In particolare, si è deciso di concentrarsi su velocità massima, accelerazione massima e decelerazione massima. SUMO, infatti, al fine di ottenere comportamenti maggiormente realistici definisce anche altre caratteristiche come ad esempio il tasso di impazienza per i guidatori. Tale attributo, a default vale 0.00 e cresce con il tempo che un automobilista è stato fermo in fila determinando la sua tolleranza ad attendere ancora oppure a commettere qualche imprudenza pur di passare, come ad esempio, non dare la precedenza a veicoli prioritari. Purtroppo però questa caratteristica sarebbe estremamente difficile da estrarre nella realtà e dunque per quanto fosse pertinente non è stata usata nella determinazione degli stili di guida.

Infine per la probabilità di generazione delle varie tipologie, si è scelto di mantenere tutte le classi di veicoli equiprobabili in quanto così si sarebbe potuta determinare più facilmente l'efficacia o meno del clustering sulla tipologia.

Questo alto livello di granularità è stato alla base del clustering supervisionato realizzato in seguito. In particolare, le simulazioni sono state svolte andando a variare di volta in volta le tipologie dei veicoli coinvolti e dunque le loro caratteristiche realizzando scenari a difficoltà crescente per il clustering.

2.3 La simulazione

Per poter eseguire una simulazione, è necessario prima popolare la rete definendo quella che in SUMO è chiamata Traffic Demand e che in altre parole non è altro che un file dove sono specificati quanti veicoli dovranno essere inseriti nella rete e con quali percorsi.

SUMO fornisce diversi modi per realizzare una Traffic Demand e tra questi vi è anche la possibilità di usare uno script Python incluso nella release che permette di generare un certo numero di percorsi in maniera totalmente casuale, sia da un punto di vista della lunghezza che della scelta dei nodi da attraversare. Tale script però, nella sua semplicità, presenta anche delle problematiche legate al fatto che la generazione dei percorsi è del tutto casuale. In particolare, si ha che:

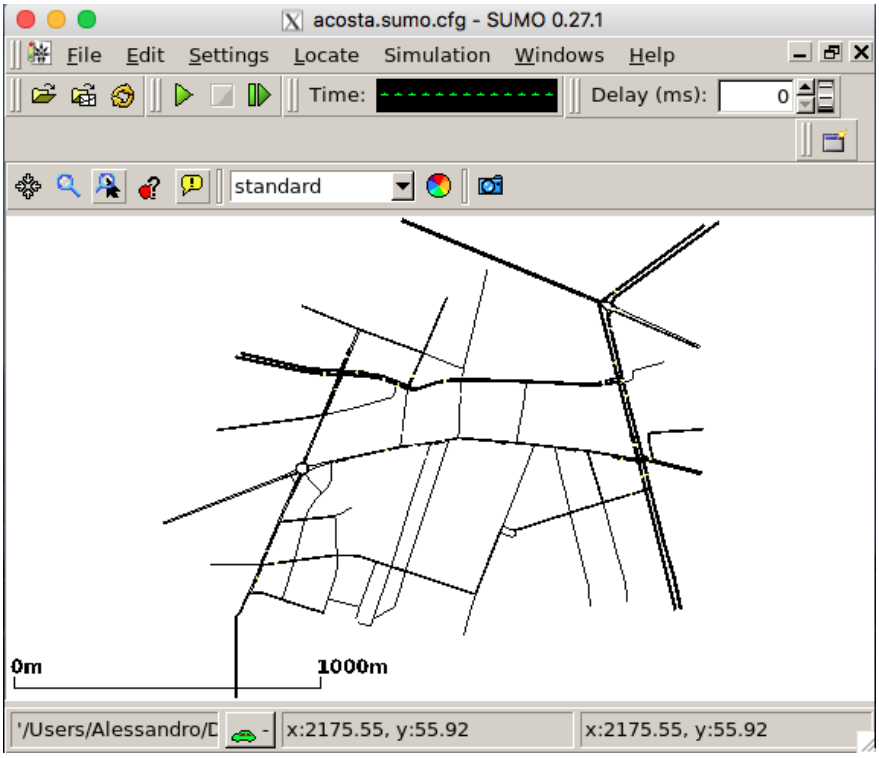
1. In reti reali non tutti i percorsi saranno validi in quanto vi potranno essere percorsi che cercano di collegare edge non adiacenti. Ciò comporta che sarà dunque necessario definirne un numero maggiore per ottenere il numero desiderato. Inoltre ciò si concretizza anche in istanti di tempo all'interno della simulazione in cui non partirà alcun veicolo

2. La generazione casuale comporta un traffico nella rete poco realistico che non tiene conto delle effettive strade che sceglierebbe un automobilista reale e che potrebbe comportare anche percorsi poco significativi che ripropongono in ciclo gli stessi edge

3. Sempre la generazione casuale comporta che vi potranno essere percorsi molto più lunghi di altri con conseguente differenza nelle tracce raccolte per i veicoli in termini di quantità

Una volta generata la Traffic Demand si può procedere con la simulazione della rete. In particolare, SUMO prevede a questo fine la possibilità di eseguire due tipologie di simulazione. In primo luogo è possibile eseguire una simulazione da riga di comando. Tale soluzione rappresenta quella più usata e anche la più adatta per reti molto grandi e articolate. In alternativa, è possibile usare la SUMO-GUI (Figura 2.4) attraverso la quale è possibile osservare graficamente il percorso di ogni singola macchina e individuare facilmente i tratti di strada più trafficati.

Figura 2.4: Screenshot sumo-gui



2.4 Gli output di una simulazione

SUMO permette di generare diverse tipologie di output a seconda di quelle che possono essere le esigenze dell'utilizzatore. In particolare, ogni tipologia colleziona valori differenti rispetto alle altre occupandosi poi di scriverli su file, tipicamente in formato XML, o su una socket. A default si ha che tutti gli output sono disabilitati e, se necessari, devono essere abilitati in modo individuale. Ciò può essere fatto direttamente da linea di comando per alcuni, mentre per altri deve essere fatto tramite additional files. Nello specifico, agli scopi del clustering, si è vista inizialmente l'opzione FCD (Floating Car Data) che rappresenta un output molto completo e che descrive una serie di caratteristiche per ogni singolo veicolo per ogni step della simulazione permettendoci di avere dei valori istantanei (Vedi tabella 2.1).

Inizialmente, in una fase prevalentemente esplorativa, si è realizzato un parser ad hoc che si occupasse di fare il parsing del file di output della simulazione SUMO FCD estraendo per ogni veicolo una serie di informazioni rilevanti quali: tipo di veicolo, velocità e accelerazione media, velocità massima, istante di ingresso e lane attraversate. Inoltre, tale parser

si occupava anche di raccogliere in forma aggregata per tipo di veicolo la velocità media, l'accelerazione media e la velocità massima riscontrata.

Tabella 2.1: Tabella Floating Car Data (FCD) output

Name	Type	Description
`timestep`	(simulation) seconds	The time step described by the values within this timestep-element
`id`	id	The id of the vehicle
`type`	id	The name of the vehicle type
`speed`	m/s	The speed of the vehicle
`angle`	degree	The angle of the vehicle
`x`	---	The absolute X coordinate of the vehicle (center of front bumper). The value depends on the given geographic projection
`y`	---	The absolute Y coordinate of the vehicle (center of front bumper). The value depends on the given geographic projection
`pos`	---	The running position (in m) of the vehicle measured from the start of the current lane.
`lane`	---	The id of the current lane.
`slope`	---	The slope (in degree) of the vehicle (equals the slope of the road at the current position)

Questo approccio presentava però due problemi principali: un problema di tempo e un problema di spazio occupato per i file xml prodotti da SUMO. Si è infatti visto che, al crescere dei tempi simulati e del numero dei veicoli, i file xml esplodevano con dimensioni nell'ordine di qualche GB e i tempi di parsing si aggiravano in un ordine di grandezza di decine di minuti. Si è dunque cercato una soluzione alternativa e la si è trovata in uno strumento offerto da SUMO noto come Traci (Traffic Control Interface).

2.5 Traci

Traci, come anticipato nella sezione 1.2 sui simulatori microscopici, è uno strumento offerto da SUMO che permette di accedere agli oggetti di una simulazione in stato di running e di manipolarne il comportamento o semplicemente estrarne le caratteristiche. Si ha che in questo modo, lo sviluppatore può estrarre solo i dati di suo interesse, in modo diretto e salvarli nel formato che preferisce. Si sono dunque realizzati degli script che potessero sfruttare le API di Traci al fine di avviare una simulazione e di estrarne in tempo reale le informazioni considerate rilevanti.

In particolare, a questo scopo si sono realizzate due modalità di simulazione:

1. Eager –> A partire dal timestep 0 sono considerati tutti i veicoli che entrano ed escono dalla simulazione. Tale modalità ha senso in combinazione con simulazioni lunghe nelle quali entrano molti veicoli, ma in modo sparso nel tempo. In particolare, le serie temporali estratte in questa modalità dovranno poi essere uniformate a posteriori in quanto avranno delle lunghezze molto diverse

2. Lazy –> Fino a quando non sono entrati quasi tutti i veicoli, non si comincia a tenere traccia delle loro caratteristiche. Questo comportamento è raggiunto usando una particolare funzione di Traci che permette di ottenere per ogni istante di tempo la lista dei veicoli in partenza. Quando tale insieme è vuoto significa che più o meno tutti i veicoli sono entrati nella simulazione. Tale modalità ha senso per simulazioni nelle quali tutti i veicoli entrano rapidamente all'interno della simulazione. In particolare, a differenza della modalità eager, le serie temporali estratte in questa modalità necessiteranno di uno sforzo inferiore di uniformazione

Nonostante l'utilizzo di Traci abbia permesso di ridurre notevolmente i tempi di simulazione, una cosa che si è potuta osservare nel corso di questa tesi è che simulare è comunque costoso in termini di tempo e di memoria. Da qui la strategia di salvare i dati estratti in un file in formato json così da poter separare la simulazione dal clustering e da permettere il riutilizzo degli stessi dati per un maggior numero di volte e per sperimentazioni diverse. Infine, un'ulteriore problematica con la quale ci si è dovuti scontrare è stata relativa al salvataggio stesso dei veicoli in formato json. Si ha infatti che, per simulazioni grandi, il dump in memoria dell'intero dizionario dei veicoli provocava un errore di memoria. Si è dunque provveduto alla scrittura degli elementi in modo singolo.

Capitolo 3

Clustering di serie temporali

3.1 Panoramica degli algoritmi di clustering

Il clustering consiste nel cercare di raggruppare un insieme di oggetti in modo tale che quelli dello stesso gruppo o cluster siano più "simili" tra loro rispetto a quelli appartenenti ad altri gruppi o clusters [17]. L'idea è dunque quella di cercare di massimizzare la differenza inter-cluster minimizzando allo stesso tempo quella intra-cluster.

Data la arbitrarietà del concetto di cluster, si ha che non esiste un unico algoritmo per fare clustering, ma ne esistono svariati, ognuno con la propria nozione di che cosa può essere definito cluster e di come trovarlo in modo efficiente. Tra le principali tipologie di algoritmi abbiamo:

1. Algoritmi partizionali: tali algoritmi per definire l'appartenenza ad un gruppo utilizzano la distanza da un punto rappresentativo del cluster (centroide, medioide ecc...)

2. Algoritmi gerarchici: tali algoritmi possono essere a loro volta agglomerativi oppure divisivi a seconda che procedano bottom up oppure top down. In generale, l'idea alla base è quella di costruire una gerarchia di partizioni caratterizzate da un numero decrescente nel primo caso (inizialmente ogni punto è un cluster), crescente nel secondo (inizialmente si ha un unico cluster)

3. Algoritmi basati su densità: in tali algoritmo un cluster è definito come un'area a maggiore densità rispetto al resto del dataset. In altre parole, fissata una densità di

soglia e un raggio visuale, un punto appartiene ad un cluster se ha nel suo raggio visuale un numero di punti maggiore della soglia fissata.

In questo lavoro di tesi è stato utilizzato l'algoritmo k-means, uno dei più semplici ed efficienti metodi di clustering, ancora largamente utilizzato. Una comparazione di algoritmi diversi di clustering è lasciata come possibile sviluppo futuro.

3.2 L'algoritmo k-means

L'algoritmo k-means è un algoritmo di clustering che permette di suddividere un insieme di oggetti in K gruppi sulla base dei loro attributi. In particolare, l'idea di k-means è che gli attributi degli oggetti possano essere rappresentati come vettori noti come feature vectors.

Fondamentalmente, l'algoritmo si compone di 3 passi. Il primo passo chiede all'utente il numero di cluster da realizzare K e individua K elementi (in modo random nella sua versione più semplice oppure usando delle euristiche nella sua versione più complessa) appartenenti al dataset. Tali elementi saranno considerati come i centroidi di partenza. Dopo questo step di inizializzazione, k-means consiste nell'iterare in modo alterno altri due ulteriori passi:

1. Assegnare ad ogni elemento del dataset al suo centroide più vicino

2. Definire dei nuovi centroidi a partire dal valore medio di tutti gli elementi del dataset assegnati al precedente centroide.

In altre parole, dato l'insieme delle osservazioni $(x_1, x_2, \ldots, x_n)$, dove ognuna di esse è un vettore d-dimensionale, e definito K dal passo di inizializzazione, si avranno K insiemi $(S_1, S_2, ..., S_k)$. k-means cerca dunque di minimizzare l'errore quadratico, ovvero la somma dei quadrati delle distanze tra ciascuna delle osservazioni e il centroide del proprio cluster. In termini matematici, il suo obiettivo è trovare:

$$argmin_s \sum_{i=1}^{k} \sum_{x \in S_i} ||x - \mu_i||$$

Dove μ_i è la media dei punti in S_i.

Quando la differenza tra i vecchi centroidi e i nuovi è minore di una certa soglia, ovvero quando i centroidi non si muovono più in modo significativo, allora l'algoritmo termina.

Lo svantaggio principale di k-means è che bisogna definire a priori il numero di cluster che si vuole rendendo dunque il risultato finale strettamente dipendente dal valore k scelto. Si ha inoltre che, sebbene sia garantita la convergenza, la qualità del risultato dipende anche da come vengono scelti i centroidi di partenza. In particolare, in questa tesi, si è usata l'inizializzazione k-means++ che cerca di identificare elementi che siano distanti fra loro nello spazio. È stato infatti provato di come questo tipo di inizializzazione conduca a risultati migliori rispetto ad un'inizializzazione casuale [18]. Infine, si è utilizzata la possibilità offerta dall'implementazione usata, di eseguire più volte l'inizializzazione al fine di scegliere quella migliore secondo alcuni criteri.

3.3 Uso di k-means in Scikit-Learn

In questo progetto si è scelto di usare una libreria python per il machine learning nota come scikit-learn [19]. In particolare, tale libreria offre una classe k-means che permette di realizzare un clustering in modo veloce e pratico semplicemente inizializzando un oggetto k-means con il numero di cluster che si vogliono realizzare. In seguito è sufficiente invocare il metodo fit passandogli come parametro la serie di osservazioni che si desidera clusterizzare come si può osservare in figura 3.1.

Figura 3.1: Dettaglio uso di k-means

```
def __cluster(self, n_clusters):
    start_time = timeit.default_timer()

    kmeans = KMeans(n_clusters=n_clusters, n_init=30, max_iter=500, random_state=42)
    kmeans.fit(self._inputs)

    self._clustering_time = timeit.default_timer() - start_time
    self._kmeans_labels = kmeans.labels_
    self._kmeans_centroids = kmeans.cluster_centers_
    self._kmeans_inertia = kmeans.inertia_
```

Una volta eseguito il clustering, poi, l'oggetto k-means esporrà attraverso opportuni attributi le label e i centroidi calcolati per il dataset considerato come anche l'inerzia finale.

3.4 Le modalità di clustering

In questo progetto, il clustering è stato eseguito attraverso lo script Cluster.py ed è stato realizzato in due modalità diverse:

1. Clustering su dati come sequenze temporali –> In tale modalità si è utilizzato come feature vector per ogni veicolo un vettore contenente le sue informazioni caratteristiche sotto forma di sequenze temporali come ad esempio la serie delle sue velocità e accelerazioni istantanee.

2. Clustering su dati aggregati –> In tale modalità si è utilizzato come feature vector per ogni veicolo un vettore contenente le sue informazioni caratteristiche aggregate tramite funzioni come ad esempio media, mediana, deviazione standard, massimo...

Nelle prossime sezioni andremo a vedere nel dettaglio le due modalità sopra descritte concentrandoci sulle problematiche incontrate e di come sono state risolte.

3.5 Clustering su dati come sequenze temporali

L'obiettivo dell'algoritmo di clustering impiegato in questo lavoro di tesi era quello di identificare classi di veicoli, corrispondenti a stili di guida comuni o frequenti, a partire da tracce che descrivessero il comportamento di ciascun veicolo nel tempo. In particolare, a tale proposito ci si è scontrati dapprima con il problema dell'eterogeneità in termini di dimensione dei feature vectors dei singoli veicoli. In seguito, un'altra problematica è stata quella relativa alla scelta di quali fossero le tracce più indicate ai fini del clustering e se fosse più giusto ordinarle in senso crescente, decrescente oppure non ordinarle affatto.

3.5.1 Le dimensioni dei feature vectors

Da documentazione skscit-learn [20], l'algoritmo di clustering k-means riceve in input una matrice $N \times M$ dove N nel nostro particolare caso rappresenta il numero di veicoli, mentre M rappresenta la lunghezza della serie di tracce scelte come rappresentative. Il problema, dunque, è stato fare in modo che M fosse uguale per tutti i veicoli presi in analisi al fine di poter eseguire poi il clustering. Si ha infatti che le variabili che influiscono sul tempo di

permanenza di un veicolo all'interno della rete di simulazione sono le più svariate. I veicoli, ad esempio, non solo possono avere percorsi di lunghezza differente, ma tipicamente entrano ed escono dalla rete in istanti diversi.

Per affrontare questa problematica, inizialmente si è pensato ad uno zero-padding [1] , ma l'idea è stata immediatamente scartata in quanto una tale soluzione sarebbe andata ad alterare di molto i risultati del clustering. Si è dunque pensato di tenere in considerazione solo una porzione della simulazione e all'interno di essa solo una porzione di veicoli. Questo però risolveva solo in parte il problema in quanto, come detto in precedenza, in SUMO i veicoli possono arrivare a destinazione e dunque uscire dalla simulazione prima che essa sia terminata e dunque il fatto di considerarne solo una porzione non poteva garantire l'omogeneità dei feature vectors. Le soluzioni sono dunque ricadute su due alternative per cercare di uniformare i feature vectors:

1. La lunghezza più frequente (most-frequent-length)

2. La lunghezza minima (min-length)

Durante gli esperimenti, si è però deciso di scartare la prima soluzione in quanto andava a ridurre i veicoli considerati e dunque era difficilmente comparabile sia con la soluzione min-length sia con altre simulazioni nelle quali si era usato most-frequent-length. Osserviamo, infatti, di come sebbene una simulazione potesse contare più di tremila veicoli, tipicamente le lunghezze erano così diverse tra loro che la più frequente poteva arrivare a contare approssimativamente un centinaio di elementi riducendo così drasticamente il numero di veicoli effettivamente considerati dal clustering.

La scelta è stata dunque quella di uniformare le tracce tagliando tutti i feature vectors alla lunghezza del più corto. Il problema, in questo caso, è stato che, tipicamente, tagliare alla lunghezza minima spesso significava avere delle tracce nell'ordine di grandezza delle venti unità non permettendo di eseguire un'analisi anche su vettori più lunghi.

L'idea è stata dunque quella di realizzare un filtro a posteriori che potesse, in un qualche modo, mantenere solo i veicoli con lunghezza maggiore o uguale ad una certa soglia scartando gli altri. Ciò avrebbe permesso di avere delle min-length maggiori sacrificando un certo numero di veicoli.

[1] Aggiunta di zeri fino al raggiungimento della lunghezza M desiderata.

In particolare, si è definita una funzione filter_by_length che si occupasse nell'ordine di:

1. Creare delle tuple: id veicolo, lunghezza del vettore delle sue velocità
2. Ordinare le tuple del punto 1 per lunghezza
3. Tagliare il vettore ordinato delle tuple del punto 2 in modo tale che iniziasse da un indice *start* definito come:

$$start = percentage * n\_vehicles$$

4. Eliminare dai veicoli considerati per il cluster tutti i veicoli i cui id non fossero presenti nel nuovo vettore definito al punto 3

Così facendo, è stato dunque possibile considerare un sottoinsieme di veicoli che avesse una lunghezza minima per le tracce maggiore. Tale lunghezza è stata poi resa personalizzabile nel file di configurazione clustering.json così da permettere di confrontare simulazioni con feature vectors di dimensioni diverse, ma sulla stessa quantità di veicoli. Il dettaglio della funzione è descritto in figura 3.2.

Figura 3.2: Dettaglio filter_by_length

```
def __filter_by_length(self, start_at_percentage):
    n_vehicles = len(self._vehicles_to_test)
    vehicles_length_tuples = []
    for v in self._vehicles_to_test:
        numeric_vid = float(v.vid.split("@")[1])
        n_inst_speeds = len(v.get_all_inst_speeds())
        vehicles_length_tuples.append((numeric_vid, n_inst_speeds))

    sorted_vehicles_length_tuples = sorted(vehicles_length_tuples, key=lambda vehicle: vehicle[1])

    start = int(start_at_percentage * n_vehicles)
    sorted_vehicles_length_tuples = sorted_vehicles_length_tuples[start:]

    logging.info("[Process %d] Vehicles taken into account %d / %d" % (self._pid, n_vehicles-start, n_vehicles))
    in_percentage = ((n_vehicles-start)*1. / n_vehicles*1.) * 100
    logging.info("[Process %d] Percentage: %.2f%%" % (self._pid, in_percentage))

    ids = [el[0] for el in sorted_vehicles_length_tuples]
    for v in self._vehicles_to_test[:]:
        numeric_vid = float(v.vid.split("@")[1])
        if numeric_vid not in ids:
            self._vehicles_to_test.remove(v)
```

Osserviamo di come, nonostante le tipologie dei veicoli siano state immesse nella rete in modo equiprobabile (come spiegato nel capitolo 2 sezione 2.2), scegliendo a posteriori solo un sottoinsieme del totale dei veicoli, questo comporti come effetto collaterale una presenza

non omogenea in tale porzione delle classi di veicolo definite. Osserviamo però anche di come questo contribuisca dall'altro lato a rendere ancora più realista l'analisi in quanto, con ogni probabilità, in uno scenario reale differenti stili di guida e veicoli saranno presenti in modo non uniforme.

3.5.2 La scelta dei feature vectors

Che cosa distingue uno stile di guida da un altro? Questa è una tra le domande che si è cercato di rispondere in questa tesi nel decidere quali fossero le tracce rilevanti sulle quali eseguire il clustering. Alte accelerazioni e velocità oltre il limite potrebbero essere un chiaro segnale di un guidatore aggressivo, ma anche la quantità e il tipo di frenate, il rispetto della precedenza e dei semafori o addirittura il modo in cui si affronta una curva potrebbero dare delle evidenze sullo stile di guida. In questa tesi, ci si è concentrati dunque esclusivamente su caratteristiche che fossero misurabili attraverso il simulatore, ma che soprattutto fossero anche poi riproducibili attraverso un opportuno tuning di parametri. In particolare, la scelta è dunque ricaduta su:

1. Velocità: tale parametro può essere controllato attraverso la definizione dello Speed Factor per ogni tipologia di veicolo

2. Accelerazione e decelerazione: tali parametri possono essere controllati rispettivamente attraverso la definizione di accelerazione e decelerazione massima per ogni tipologia di veicolo

In seguito, ci si è chiesti se l'espressività delle tracce estratte potesse in un qualche modo cambiare con un ordinamento. Si sono dunque svolti alcuni esperimenti in questo senso, anche combinando tra loro diverse tipologie di tracce.

3.6 Clustering su dati aggregati

Come alternativa all'utilizzo delle intere serie temporali come vettori di feature, si sono utilizzati anche dati aggregati estratti da ciascuna serie. In particolare, in questa tipologia di clustering non ci si è scontrati con il problema della dimensionalità in quanto, utilizzando i dati in forma aggregata, la dimensione dei feature vectors era la stessa per tutti i veicoli.

Anche qui la difficoltà è stata nella scelta di quali caratteristiche aggregare e le considerazioni sono state le medesime che sono state fatte per i dati sequenziali. Ci si è poi anche chiesti che tipo di aggregazione fare sulle tracce orientandosi infine su quelle più comuni di: Massimo, deviazione standard, media e mediana.

3.7 L'analisi del clustering

Una volta eseguito il clustering, per valutarne la qualità si sono utilizzati indici quantitativi e grafici qualitativi che ne attestassero l'efficacia.

3.7.1 Le metriche utilizzate

Valutare le performance di un algoritmo di clustering è un'attività complessa e vi sono diverse metriche per farlo [21]. In particolare, ai fini di questa tesi sono state prese in considerazione le seguenti:

1. Adjusted Rand Index (ARI): data la conoscenza delle labels reali e di quelle assegnate dall'algoritmo di clustering usato, tale metrica misura la similarità tra i due assegnamenti. In particolare, l'Adjusted Rand Index ignora le permutazioni, è normalizzato rispetto ad un assegnamento causale dei cluster ed è simmetrico. Il range per l'Adjusted Rand Index è tra -1 (pessimo) e 1 (ottimo)

2. Adjusted Mutual Information (AMI): data la conoscenza delle labels reali e di quelle assegnate dall'algoritmo di clustering usato, anche tale metrica misura la similarità tra i due assegnamenti. In particolare, anche l'Adjusted Mutual Information ignora le permutazioni, è normalizzato rispetto ad un assegnamento causale dei cluster ed è simmetrico. Il range per l'Adjusted Mutual Information è tra 0 (pessimo) e 1 (ottimo)

3. Omogeneità: un risultato di clustering soddisfa il criterio di omogeneità se tutti i suoi cluster contengono solamente punti che sono membri di un'unica classe [22]. Tale metrica ignora le permutazioni, ma non è simmetrica. In generale, vale la relazione:

$$homogeneity\_score(a, b) == completeness\_score(b, a)$$

Per calcolare lo score di omogeneità è necessaria la conoscenza delle labels assegnate dall'algoritmo di clustering usato. Il range per l'omogeneità è tra 0 (pessimo) e 1 (ottimo)

4. Completezza: un risultato di clustering soddisfa il criterio di completezza se tutti i i punti che sono membri di una stessa classe appartengono allo stesso cluster [23]. Tale metrica ignora le permutazioni, ma non è simmetrica. In generale, vale la relazione:

$$homogeneity\_score(a, b) == completeness\_score(b, a)$$

Per calcolare lo score di completezza è necessaria la conoscenza delle labels assegnate dall'algoritmo di clustering usato. Il range per la completezza è tra 0 (pessimo) e 1 (ottimo)

5. V-measure: rappresenta la media armonica tra omogeneità e completezza ed è dunque definita come:

$$v = 2 * \frac{h * c}{h + c}$$

Anche tale metrica ignora le permutazioni, ma a differenza di completezza e omogeneità, è simmetrica. Anche in questo caso, per calcolare lo score di V-measure è necessaria la conoscenza delle labels assegnate dall'algoritmo di clustering usato. Il range per la V-measure è tra 0 (pessimo) e 1 (ottimo)

6. Silhouette: tale metrica, a differenza delle precedenti, può essere usata quando non si conoscono le labels reali in quanto sfrutta il modello stesso per capire se il clustering è stato efficace o meno. In particolare definite:

   (a) a_i la distanza media dell'oggetto i-esimo rispetto agli altri oggetti dello stesso cluster

   (b) b_i il minimo tra le distanze medie dell'oggetto i-esimo rispetto agli oggetti degli altri cluster

Si ha che l'indice di silhouette per l'oggetto i-esimo è definito come:

$$s_i = (b_i - a_i)/max(a_i, b_i)$$

Per calcolare poi l'indice di silhouette complessivo si esegue poi una media dei singoli indici di silhouette calcolati. Il range per la silhouette è tra -1 (pessimo clustering) e 1 (ottimo clustering) con valori intorno allo zero ad indicare cluster sovrapposti

Un altro strumento che potrebbe essere usato per misurare le performance del clustering è l'inerzia di k-means o somma dei quadrati intra-cluster. Tale indicatore può essere visto come la misura di quanto siano coerenti internamente i vari cluster ed è anche ciò che k-means cerca di minimizzare ad ogni iterazione scegliendo dei nuovi centroidi.

In conclusione, si è pensato che ai fini di questo progetto fossero più rilevanti metriche che tenessero in considerazione anche le labels reali escludendo dunque dai criteri di valutazione sia l'inerzia k-means che l'indice di silhouette. Si è infatti osservato che tali indici potevano assumere valori ottimi anche nel caso in cui l'algoritmo k-means non fosse riuscito a identificare le tipologie di guidatori inserite all'interno della rete. Si pensi ad esempio al caso in cui k-means identifichi in modo inequivocabile due cluster molto densi nonostante le tipologie di automobilisti inserite manualmente fossero quattro. Ciò determinerebbe un indice di silhouette che con ogni probabilità sarebbe alto, ma senza significare che k-means abbia riconosciuto gli stili di guida inseriti.

Riepilogando dunque abbiamo usato solo gli indici mostrati in tabella 3.1.

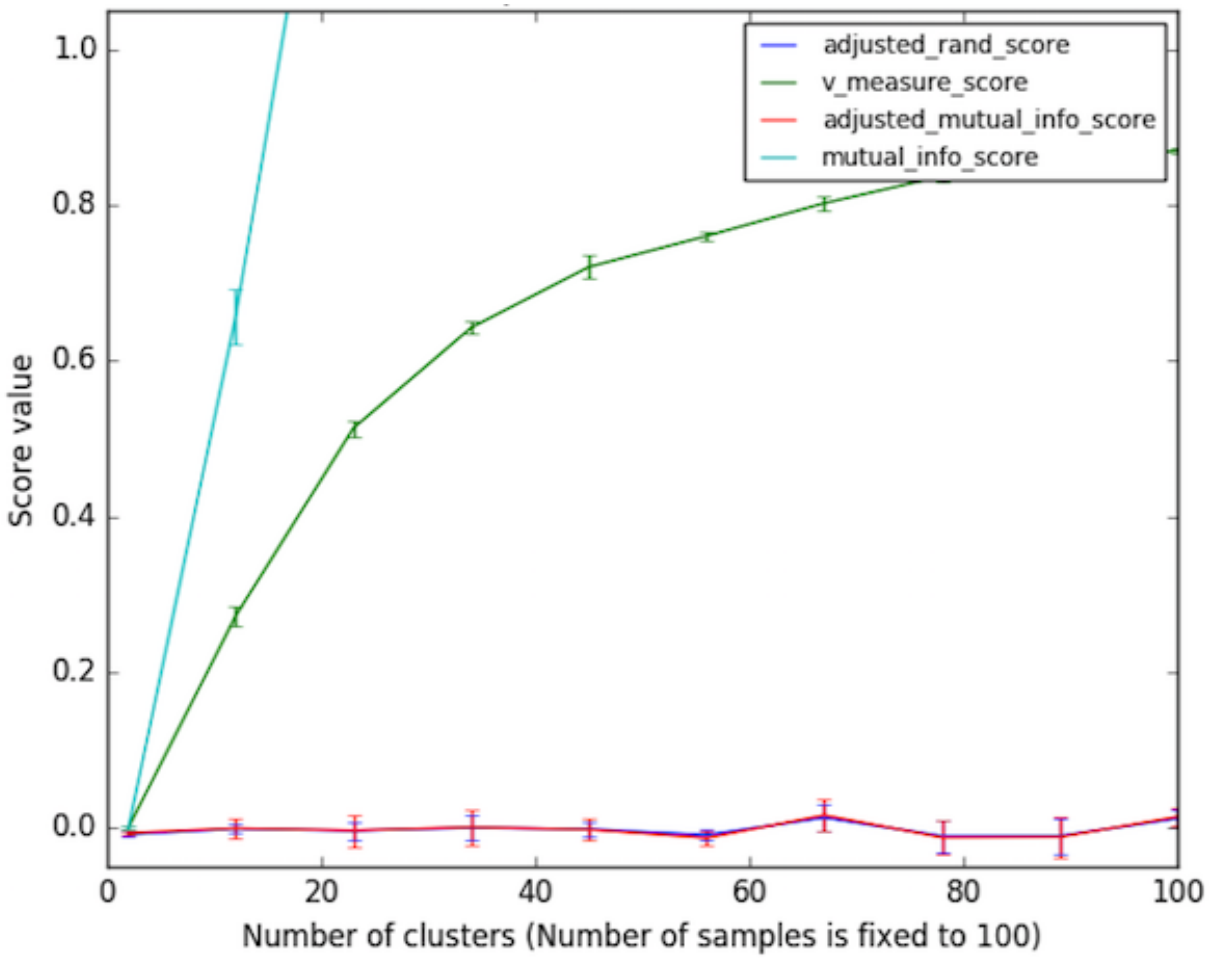
Tabella 3.1: Riepilogo delle metriche usate

Metrica	**Ignora permutazioni**	**Simmetrica**	**Normalizzata rispetto al caso**	**Range**
ARI	SI	SI	**SI**	[-1, 1]
AMI	SI	SI	**SI**	[0, 1]
Omogeneità	SI	NO	**NO**	[0, 1]
Completezza	SI	NO	**NO**	[0, 1]
V-measure	SI	SI	**NO**	[0, 1]

Un'osservazione ulteriore va fatta sulla normalizzazione rispetto ad un assegnamento causale dei cluster. Si ha infatti che le uniche metriche ad avere questa caratteristica sono l'Adjusted Rand Index e l'Adjusted Mutual Information. Ciò significa che per quel che concerne omogeneità, completezza e V-measure a seconda della dimensione del dataset analizzato, del numero di cluster e del numero di classi reali, un assegnamento casuale delle

labels non sempre potrebbe riflettersi in punteggi prossimi allo zero, specialmente quando il numero di cluster è elevato. È stato però anche dimostrato che questo problema può essere ignorato quando il numero di elementi analizzati supera le mille unità e il numero di cluster si mantiene sotto i dieci. Possiamo osservare in figura 3.3 di come il comportamento sia invece pessimo al crescere del numero di cluster con dataset piccoli.

Figura 3.3: Misure di clustering per due assegnamenti casuali a parità di cluster



Ai fini di questa tesi, però, completezza, omogeneità e V-measure sono state considerate come attendibili in quanto il numero di cluster usato è sempre stato pari a quattro e il numero di elementi sempre sopra il migliaio.

3.7.2 Rappresentazione grafica tramite riduzione dimensionale

Uno dei problemi principali che si è incontrato nella redazione dei grafici è stato sicuramente quello dell'elevata dimensionalità dei feature vectors che si voleva rappresentare, soprattutto nel caso sequenziale. La soluzione che si è trovata è stata quella di usare la riduzione dimensionale. Dataset ad alta dimensionalità possono essere molto complicati da visualizzare. Mentre dati in due o tre dimensioni, infatti, possono essere facilmente graficati al fine di mostrare la relazione tra i dati, equivalenti grafici in maggiori dimensioni possono essere molto meno intuitivi. Per facilitare dunque la visualizzazione dei dati, la dimensione degli stessi deve essere opportunamente ridotta in qualche modo. Il modo più semplice per raggiungere questo obiettivo è sicuramente quello di eseguire una proiezione randomica dei dati. Osserviamo però di come questo approccio, per quanto semplice, rischi di perdere interessanti relazioni che potrebbero esserci fra i dati. Al fine di superare questo problema sono stati creati una serie di strumenti, supervisionati e non, come ad esempio l'analisi delle componenti principali (PCA) o delle componenti indipendenti. Questi algoritmi definiscono dei metodi per scegliere una proiezione lineare dei dati che sia "interessante". Tali metodi possono essere molto potenti, ma spesso perdono comunque alcune relazioni non lineari presenti fra i dati. Sebbene esista una branca del Machine Learning nota come Manifold Learning che tenta di generalizzare degli strumenti lineari come PCA in modo da renderli in grado di mantenere le relazioni non lineari tra i dati [24], in questo progetto si sono ritenute sufficienti i metodi di riduzione lineari. In particolare, si sono utilizzate le due rappresentazioni menzionate: Random projection e Principal Component Analysis (PCA).

Capitolo 4

Le simulazioni eseguite

4.1 Modalità di simulazione

Ai fini di questo progetto, sono state eseguite diverse simulazioni per poter provare un numero di veicoli che fosse il più eterogeneo possibile sia in termini di quantità che di tipologia. In particolare, sono state simulate quattro tipologie di veicoli (Class1, Class2, Class3 e Class4) in tre differenti scenari a difficoltà crescente:

1. Scenario "All different" o scenario semplice, nel quale i veicoli differiscono sia per caratteristiche fisiche, come ad esempio accelerazione e decelerazione, sia per speed factor, ovvero tendenza a rispettare i limiti di velocità o meno

2. Scenario "Increasing sf" o scenario intermedio, nel quale i veicoli differisco solo per aggressività, ovvero per speed factor. In particolare, in questa modalità si è pensato di simulare uno stesso veicolo, ma con un'aggressività crescente

3. Scenario "Merged" o scenario difficile, nel quale le classi di veicoli che hanno le stesse caratteristiche fisiche, hanno una differente aggressività.

Si è deciso invece di mantenere la speedDev pari a 0.1 al fine di avere veicoli il più omogenei possibile all'interno della stessa classe.

Ai fini dell'analisi, si è scelto di usare una simulazione di 14400 secondi (quattro ore) con una portata di circa 3600 veicoli all'ora. I veicoli sono stati poi filtrati a posteriori tramite la funzione definita nel capitolo 3 sezione 3.5.1 selezionandone circa il 25%. Questo, come anticipato, ha permesso da un lato di confrontare feature vectors di lunghezza differente e

dall'altro ha contribuito a rendere l'analisi più realistica andando ad alterare le proporzioni delle varie classi di veicoli.

Ogni scenario è stato analizzato con:

1. Dati aggregati
2. Serie temporali delle sole velocità
3. Serie temporali delle sole accelerazioni
4. Serie temporale delle coppie velocità e accelerazione.

Le serie temporali, poi, sono state analizzate in tre casi distinti: (i) senza ordinamento (ii) con ordinamento crescente (iii) con ordinamento decrescente.

I feature vectors dei dati aggregati consistono in tuple di tre valori: velocità massima, accelerazione massima e decelerazione massima. Dal momento che, come abbiamo visto nella sezione 2.2, questi sono proprio i parametri che SUMO permette di configurare per la definizione delle singole tipologie di veicoli, ci si aspetta un clustering molto efficace e che in un certo senso offra una sorta di limite superiore per tutti gli altri approcci. Nel seguito della trattazione andremo dunque a dettagliare per ogni scenario i risultati che ne sono derivati concentrandoci prevalentemente sull'analisi dei dati sequenziali e utilizzando invece l'analisi dei rispettivi dati aggregati come termine di confronto.

Tutti i test e le simulazioni sono stati condotti utilizzando sumo versione 0.27.1.

Capitolo 5

Risultati sperimentali scenario semplice

5.1 I valori utilizzati

Nello scenario All different, come anticipato nella sezione precedente, i veicoli differiscono sia per caratteristiche fisiche sia per speed factor. In particolare, i valori scelti sono elencati in tabella 5.1.

Tabella 5.1: Valori utilizzati per le classi di veicoli nello scenario semplice

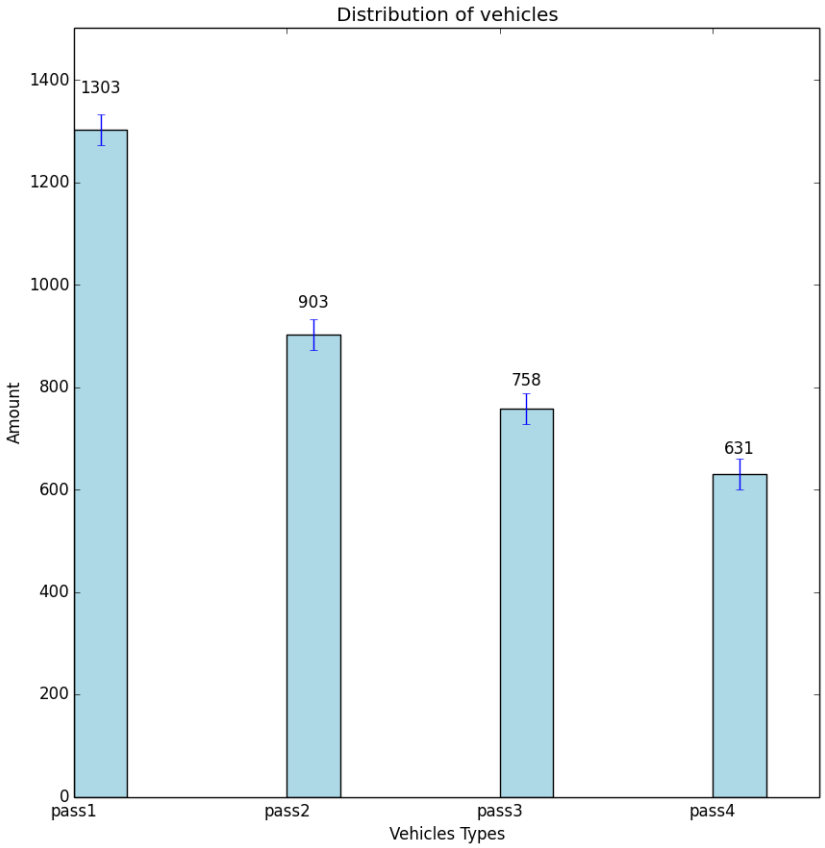
id	**accel**	**decel**	**speedFactor**
Class1	1.6	4.0	0.8
Class2	2.2	4.5	1.0
Class3	2.8	5.0	1.2
Class4	3.4	5.5	1.4

Tale scenario si configura come quello più semplice in quanto le classi di veicoli sono differenziate al massimo.

5.2 Analisi dei dati sequenziali

Per quel che concerne i dati sequenziali, l'applicazione del filtro a posteriori ha determinato una distribuzione disomogenea delle classi di veicoli con una prevalenza di Class1 (36.2%), seguiti nell'ordine da Class2 (25.1%), Class3 (21.1%) e Class4 (17.6%). Tale distribuzione è meglio rappresentata in figura 5.1.

Figura 5.1: Distribuzione delle classi di veicoli all'interno del sottoinsieme analizzato per lo scenario semplice



Nel seguito si delineeranno i risultati ottenuti nei tre casi di ordinamento menzionati nel capitolo precedente, ovvero nel caso di feature vectors non ordinati, ordinati in senso crescente e in senso decrescente. Infine si eseguirà un confronto fra i risultati migliori ottenuti con il caso dei dati aggregati.

5.2.1 Feature vectors senza ordinamento

Per quel che concerne i risultati ottenuti per feature vectors senza ordinamento, quello che si è osservato è stato di come le metriche assumessero valori migliori in modo inversamente proporzionale alla lunghezza dei feature vectors presi in esame. Si è infatti visto come, aumentando la dimensione di quest'ultimi, k-means avesse maggiori difficoltà nel riconoscere delle tipologie di veicoli all'interno della simulazione. Viceversa, con feature vectors più corti, la rilevazione delle diverse tipologie è migliorata nettamente. Come si può osservare dalla tabella 5.2, infatti, nei casi dove la lunghezza dei feature vectors è maggiore, gli indici

assumono valori più vicini allo zero, mentre nei casi in cui la lunghezza è inferiore il valore degli indici aumenta raggiungendo dei picchi di 0.6 / 0.7 per feature vectors di 10 unità.

Tabella 5.2: Confronto delle metriche al variare della dimensione dei feature vectors (fv_length) - Caso non ordinati

Modality: Unsorted - All different
Used vehicles: 3595

mode	fv_length	homo	completeness	v_meas	ARI	AMI
S	100	0.007	0.007	0.007	0.003	0.006
S	50	0.217	0.216	0.216	0.205	0.215
S	25	0.444	0.505	0.473	0.522	0.444
S	10	0.745	0.772	0.758	0.788	0.745
S	5	0.607	0.633	0.62	0.656	0.607
A	100	0.417	0.427	0.422	0.483	0.416
A	50	0.234	0.228	0.231	0.211	0.228
A	25	0.383	0.397	0.39	0.439	0.382
A	10	0.629	0.668	0.648	0.686	0.629
A	5	0.575	0.611	0.592	0.626	0.575
SA	100	0.23	0.23	0.23	0.243	0.229
SA	50	0.452	0.509	0.479	0.53	0.451
SA	25	0.717	0.75	0.733	0.762	0.717
SA	10	0.613	0.639	0.626	0.661	0.613
SA	5	0.397	0.425	0.411	0.425	0.396

5.2.2 Feature vectors con ordinamento crescente

Per quel che riguarda i feature vectors con ordinamento crescente i risultati ottenuti sono stati del tutto scoraggianti indipendentemente dalla lunghezza dei feature vectors utilizzata. Si ha infatti che, i dati sequenziali, se ordinati in modo crescente, hanno degli indici con valori sempre molto bassi e che, anzi, tendono a seguire un andamento opposto aumentando con il crescere delle dimensioni dei feature vectors (Tabella 5.3)

5.2.3 Feature vectors con ordinamento decrescente

Per quel che concerne infine i feature vectors con ordinamento decrescente, i risultati hanno superato di molto quelli ottenuti nei due casi precedenti. Si ha infatti che, anche

Tabella 5.3: Confronto delle metriche al variare della dimensione dei feature vectors (fv_length) - Caso ordinamento crescente

Modality: Sorted - All different
Used vehicles: 3595

mode	fv_length	homo	completeness	v_meas	ARI	AMI
S Sorted	100	0.018	0.018	0.018	-0.001	0.017
S Sorted	50	0.012	0.024	0.016	-0.019	0.011
S Sorted	25	0.006	0.032	0.01	-0.009	0.005
S Sorted	10	0.003	0.061	0.006	-0.003	0.002
S Sorted	5	0.002	0.071	0.004	-0.001	0.001
A Sorted	100	0.266	0.294	0.28	0.314	0.266
A Sorted	50	0.184	0.193	0.188	0.15	0.183
A Sorted	25	0.106	0.115	0.11	0.09	0.105
A Sorted	10	0.047	0.068	0.056	0.022	0.046
A Sorted	5	0.024	0.044	0.032	0.006	0.024
SA Sorted	100	0.003	0.003	0.003	-0.004	0.002
SA Sorted	50	0.003	0.005	0.004	-0.009	0.003
SA Sorted	25	0.055	0.069	0.061	0.075	0.054
SA Sorted	10	0.054	0.066	0.059	0.062	0.053
SA Sorted	5	0.043	0.056	0.048	0.041	0.042

in questo caso come per il caso unsorted, l'efficacia del clustering migliora con il diminuire della lunghezza dei feature vectors, ma a differenza del caso unsorted, raggiunge tipicamente valori migliori a parità di lunghezza toccando anche picchi di 0.98 (Tabella 5.4)

5.3 Confronto con i dati aggregati

Come ci si poteva aspettare, il clustering effettuato sulla base dei dati aggregati è risultato il più efficace ponendosi come limite superiore per tutti gli altri approcci. In tabella 5.5 sono mostrati i migliori risultati del clustering sui dati sequenziali a confronto con quelli ottenuti nel caso aggregato. I risultati del caso con ordinamento crescente sono stati omessi in quanto non considerati rilevanti. In figura 5.2 sono invece mostrati i grafici relativi. Seguono una serie di grafici relativi ottenuti attraverso la riduzione dimensionale che vanno a mostrare in modo evidente l'andamento dei dati sequenziali al variare delle dimensioni dei feature vectors usati nei casi senza ordinamento e con ordinamento decrescente

Tabella 5.4: Confronto delle metriche al variare della dimensione dei feature vectors (fv_length) - Caso ordinamento decrescente

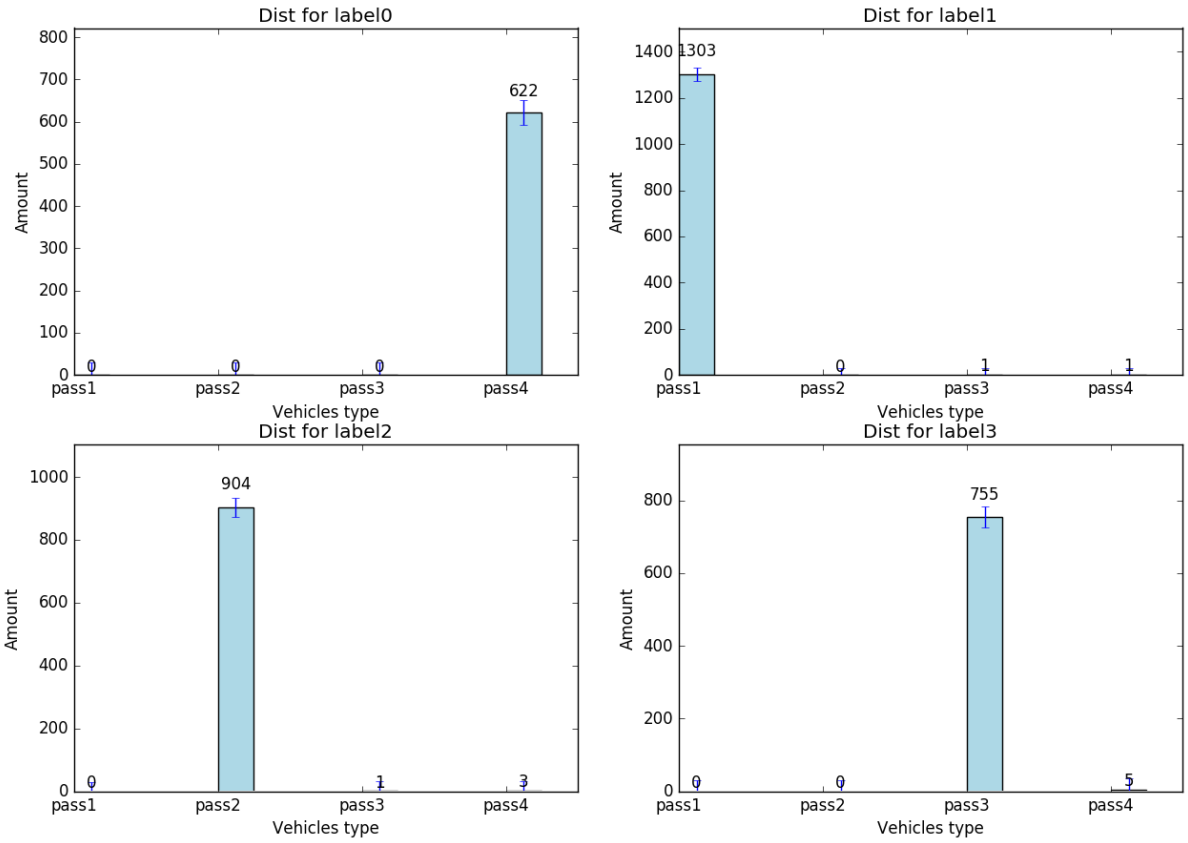
Modality: Reversed - All different
Used vehicles: 3595

mode	fv_length	homo	completeness	v_meas	ARI	AMI
S Reversed	100	0.704	0.714	0.709	0.757	0.704
S Reversed	50	0.853	0.864	0.858	0.895	0.852
S Reversed	25	0.926	0.931	0.929	0.955	0.926
S Reversed	10	0.963	0.964	0.964	0.98	0.963
S Reversed	5	0.977	0.979	0.978	0.989	0.977
A Reversed	100	0.64	0.646	0.643	0.67	0.639
A Reversed	50	0.736	0.755	0.746	0.779	0.736
A Reversed	25	0.895	0.902	0.898	0.923	0.895
A Reversed	10	0.96	0.962	0.961	0.976	0.96
A Reversed	5	0.98	0.981	0.98	0.989	0.98
SA Reversed	100	0.582	0.603	0.592	0.643	0.581
SA Reversed	50	0.671	0.706	0.688	0.731	0.67
SA Reversed	25	0.759	0.78	0.769	0.815	0.759
SA Reversed	10	0.835	0.848	0.841	0.884	0.835
SA Reversed	5	0.872	0.882	0.877	0.915	0.872

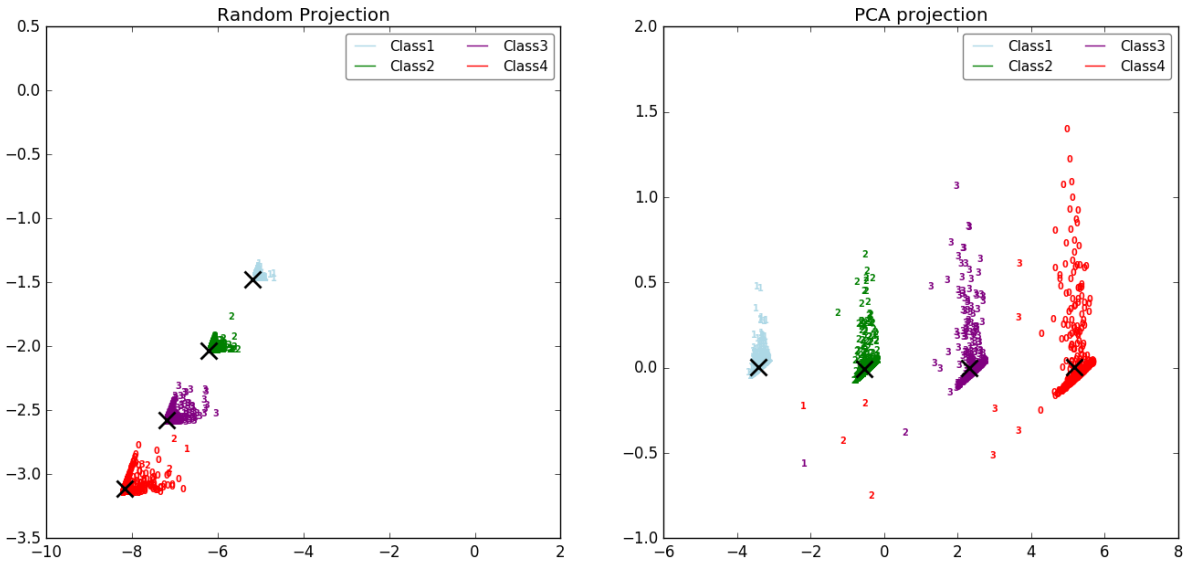
Tabella 5.5: Confronto fra i migliori risultati ottenuti nello scenario semplice: h sta per omogeneità, c per completezza, v per V-measure. Per le tracce S sta per speed, A per accelerazione e SA per la combinazione di queste due. Agg indica i risultati per le feature aggregate

Trace	Sorting	m	h	c	v	ARI	AMI
S	None	10	0.745	0.772	0.758	0.788	0.745
A	None	15	0.635	0.671	0.652	0.693	0.634
SA	None	20	0.741	0.768	0.754	0.786	0.741
S	Dec	5	0.977	0.979	0.978	0.989	0.977
A	Dec	5	0.979	0.980	0.979	0.989	0.980
SA	Dec	5	0.872	0.882	0.877	0.915	0.872
Agg	–	–	0.985	0.985	0.985	0.993	0.985

Figura 5.2: Grafici relativi alla distribuzione dei veicoli nel caso di dati aggregati

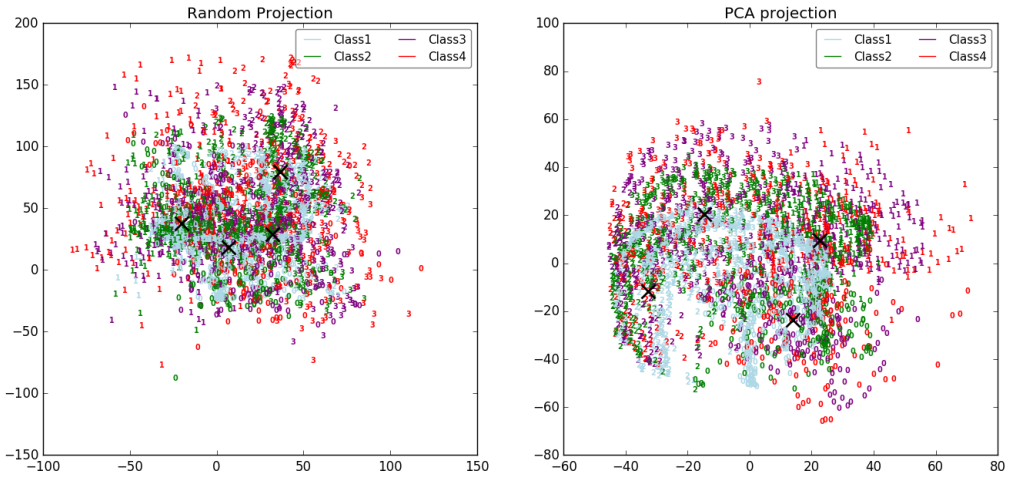


(a) Distribuzione delle label k-means nel caso aggregato

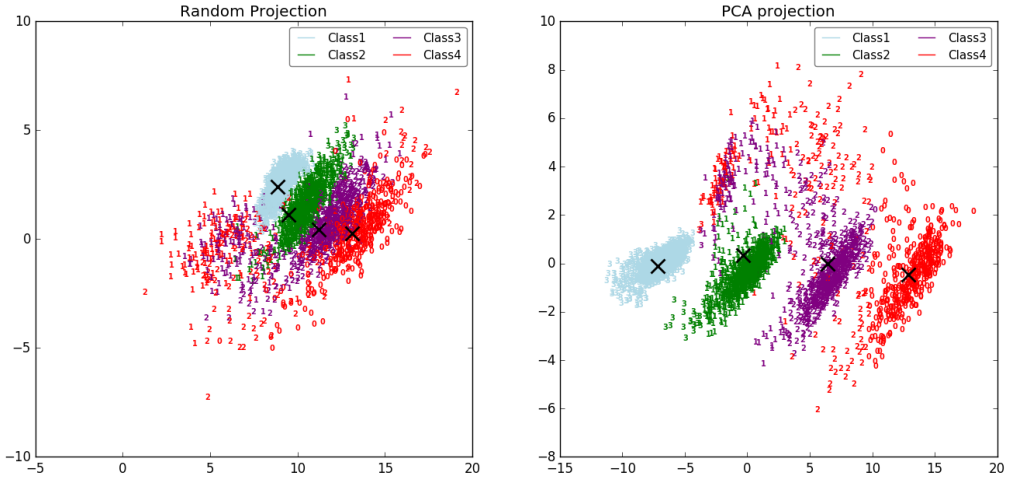


(b) Riduzione dimensionale nel caso aggregato

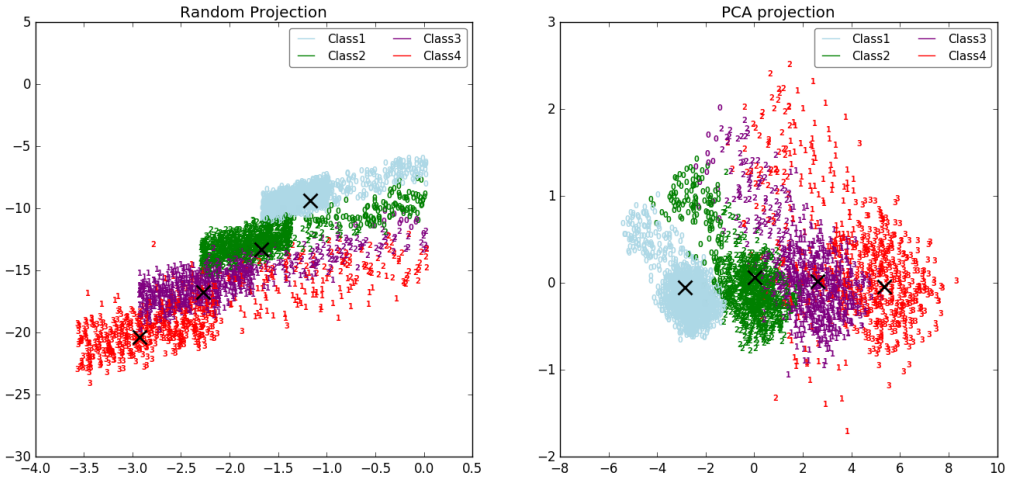
Figura 5.3: Riduzione dimensionale - Confronto fra feature vectors costituiti dalle sole velocità nel caso senza ordinamento



(a) Feature vectors: Serie delle velocità - Lunghezza: 100

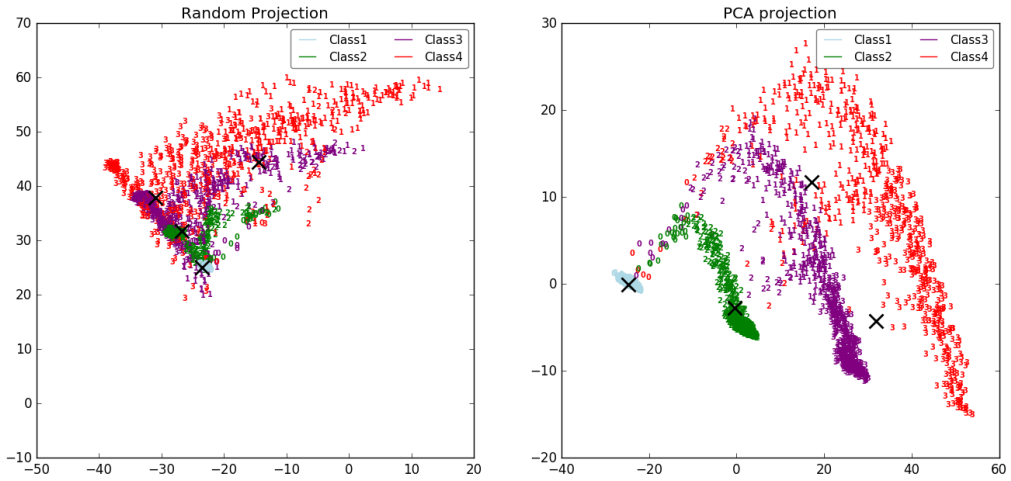


(b) Feature vectors: Serie delle velocità - Lunghezza: 10

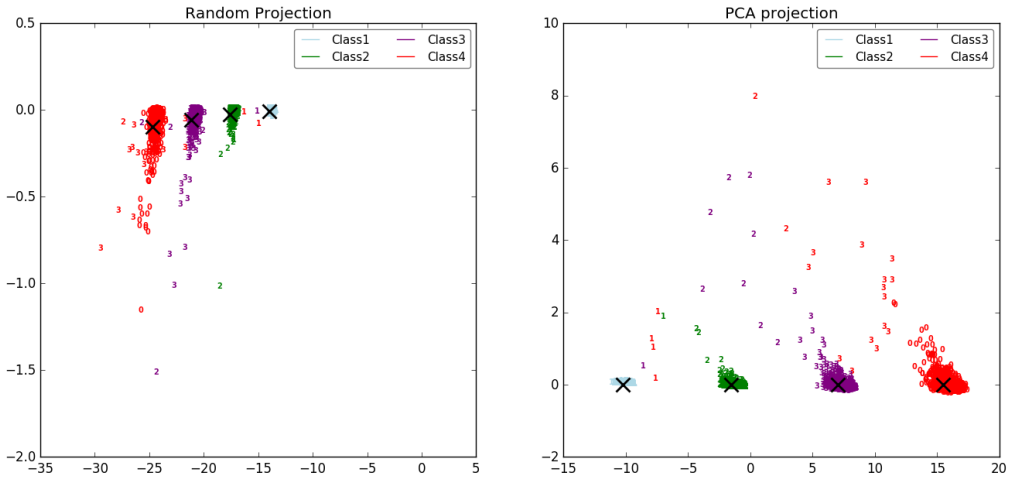


(c) Feature vectors: Serie delle velocità - Lunghezza: 5

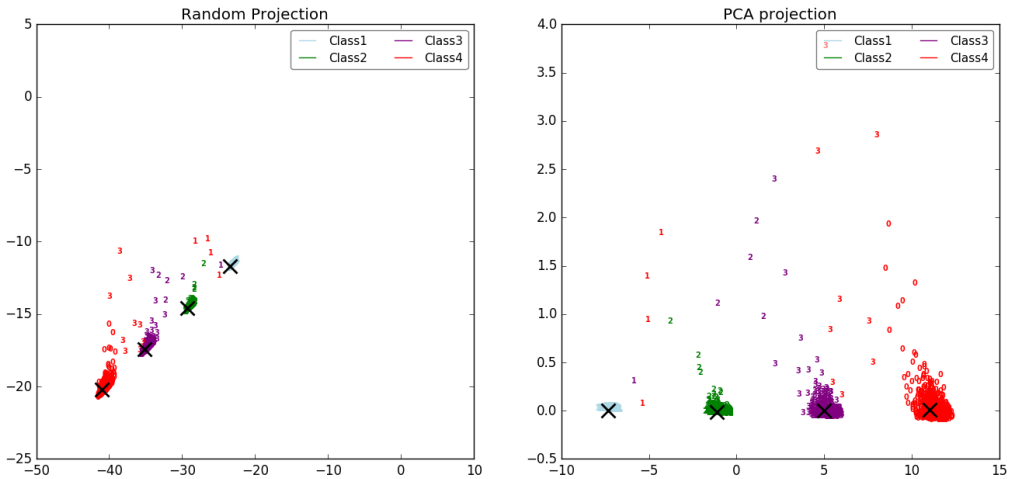
Figura 5.4: Riduzione dimensionale - Confronto fra feature vectors costituiti dalle sole velocità nel caso di ordinamento decrescente



(a) Feature vectors: Serie delle velocità - Lunghezza: 100

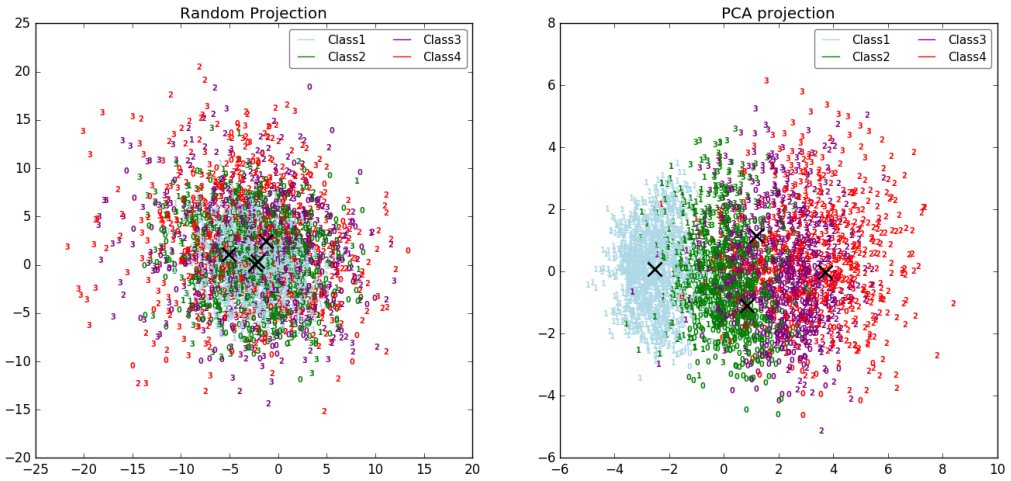


(b) Feature vectors: Serie delle velocità - Lunghezza: 10

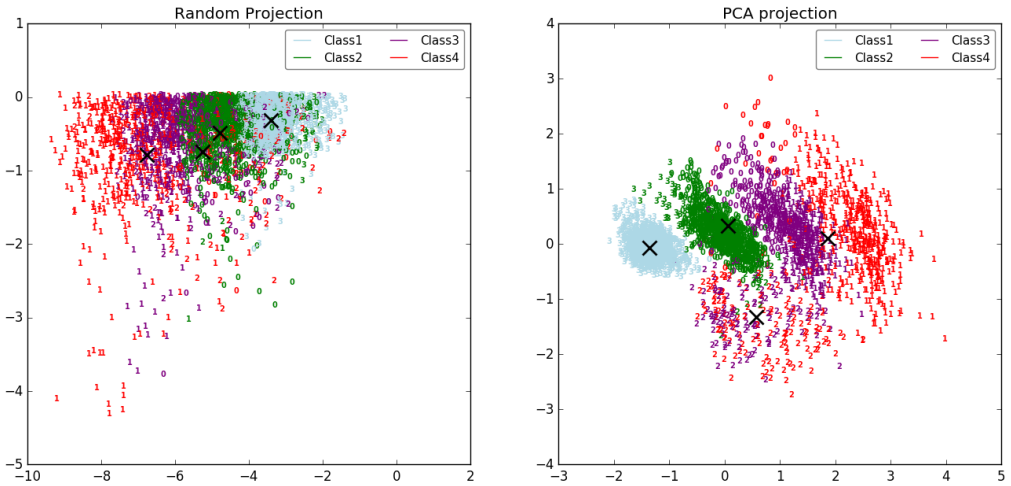


(c) Feature vectors: Serie delle velocità - Lunghezza: 5

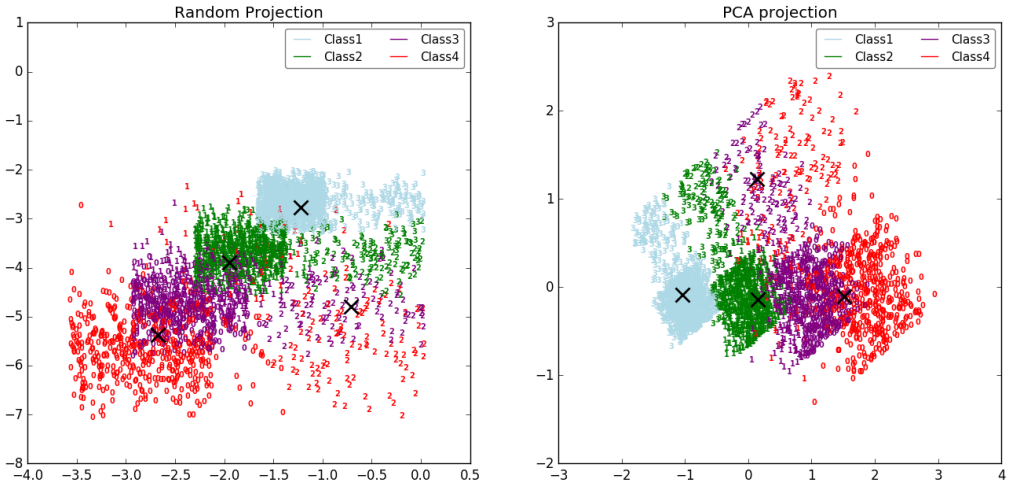
Figura 5.5: Riduzione dimensionale - Confronto fra feature vectors costituiti dalle sole accelerazioni nel caso senza ordinamento



(a) Feature vectors: Serie delle accelerazioni - Lunghezza: 100

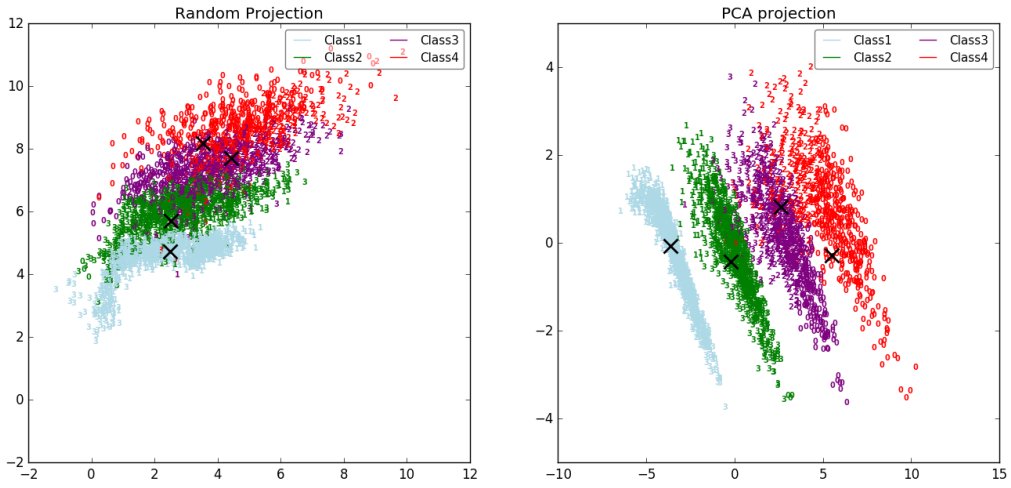


(b) Feature vectors: Serie delle accelerazioni - Lunghezza: 15

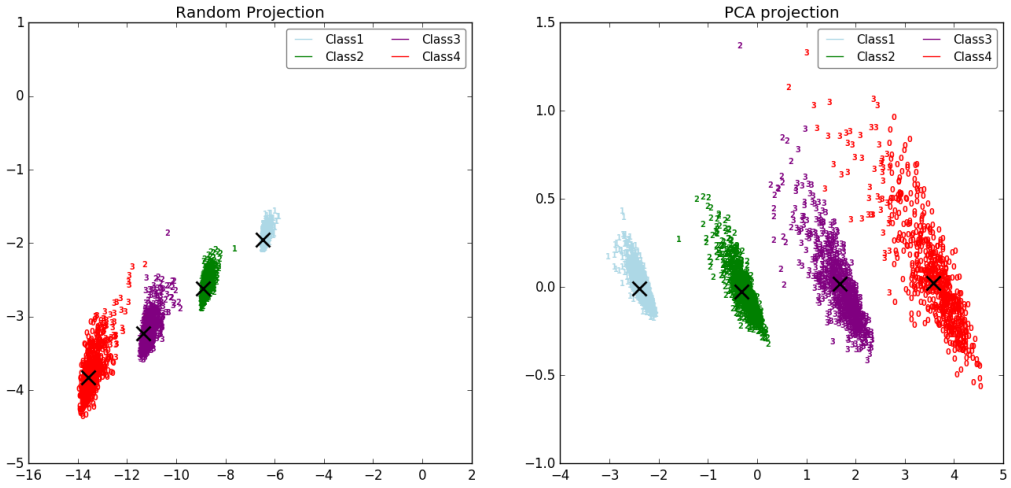


(c) Feature vectors: Serie delle accelerazioni - Lunghezza: 5

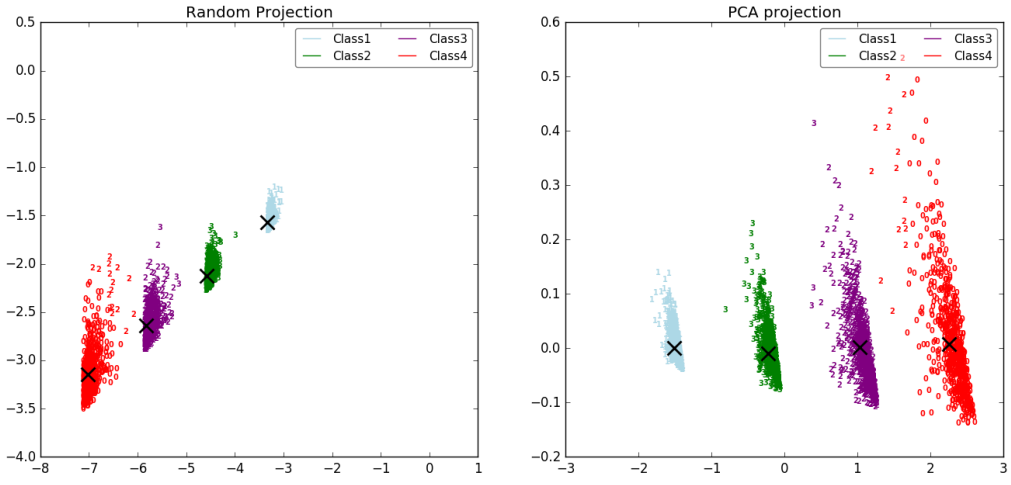
Figura 5.6: Riduzione dimensionale - Confronto fra feature vectors costituiti dalle sole accelerazioni nel caso di ordinamento decrescente



(a) Feature vectors: Serie delle accelerazioni - Lunghezza: 100

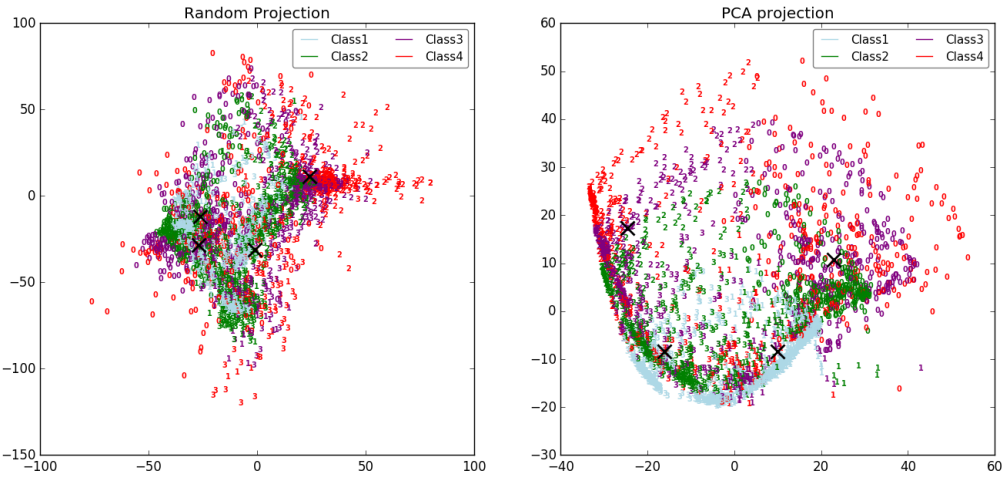


(b) Feature vectors: Serie delle accelerazioni - Lunghezza: 15

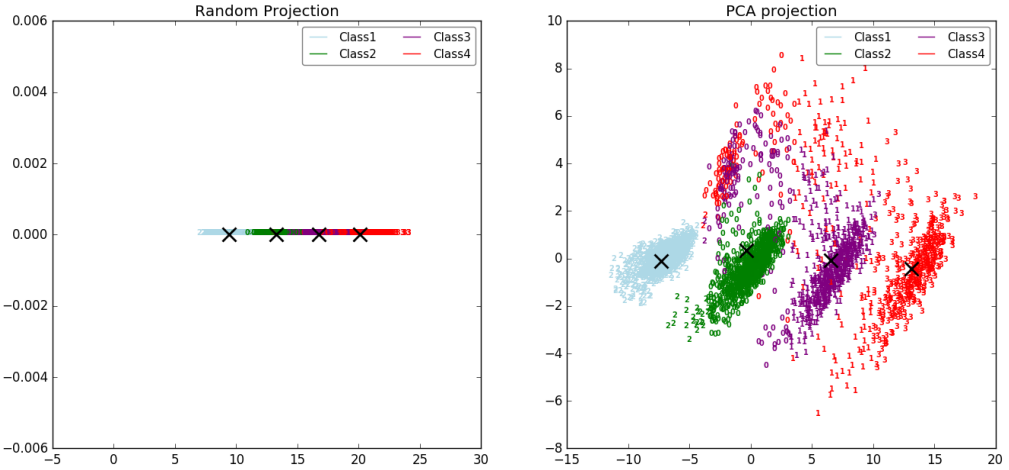


(c) Feature vectors: Serie delle accelerazioni - Lunghezza: 5

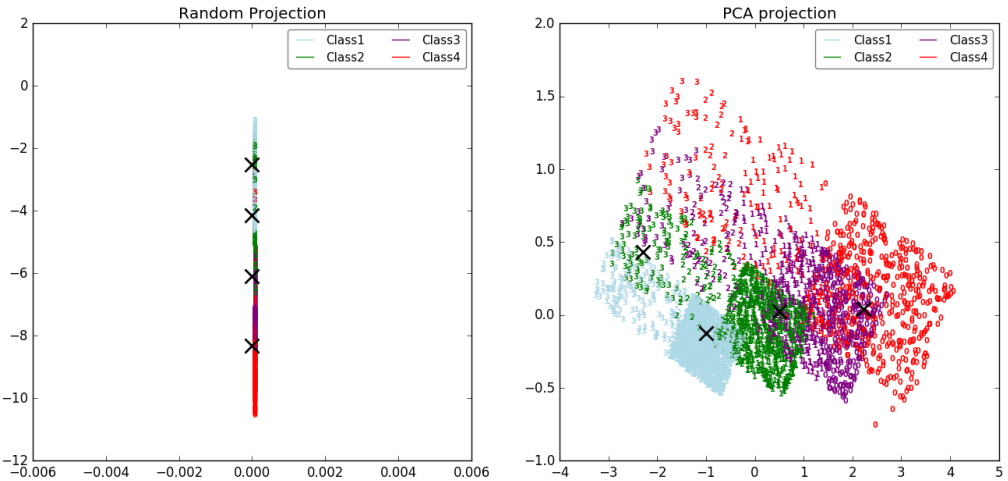
Figura 5.7: Riduzione dimensionale - Confronto fra feature vectors costituiti dalle coppie velocità e accelerazione nel caso senza ordinamento



(a) Feature vectors: Coppie velocità e accelerazione - Lunghezza: 100

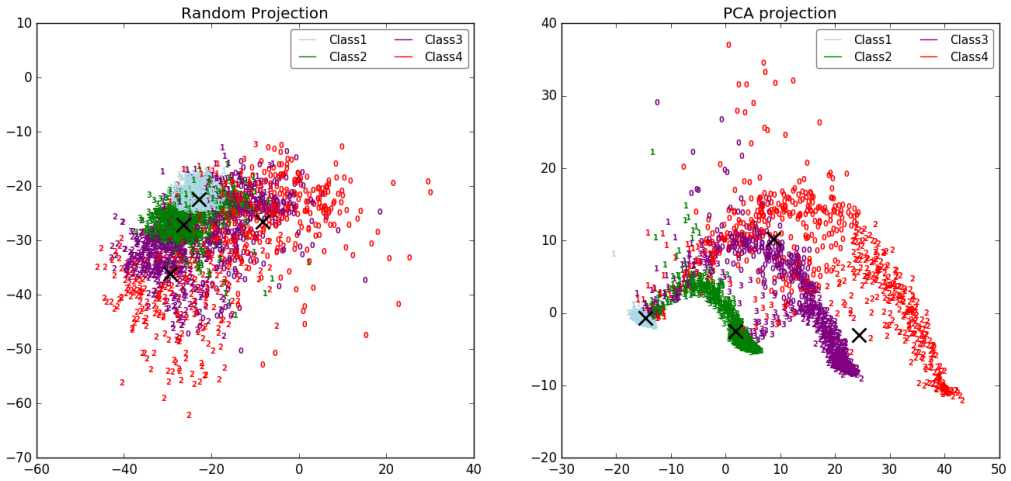


(b) Feature vectors: Coppie velocità e accelerazione - Lunghezza: 20

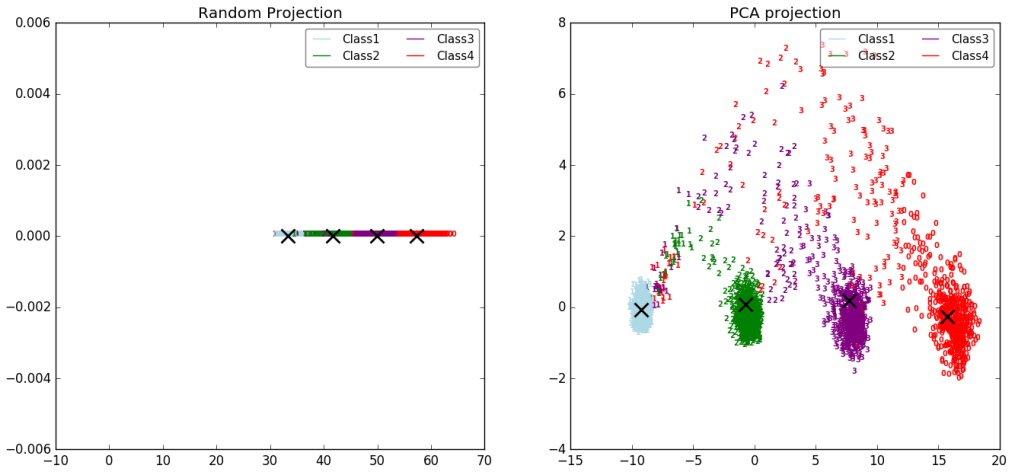


(c) Feature vectors: Coppie velocità e accelerazione - Lunghezza: 5

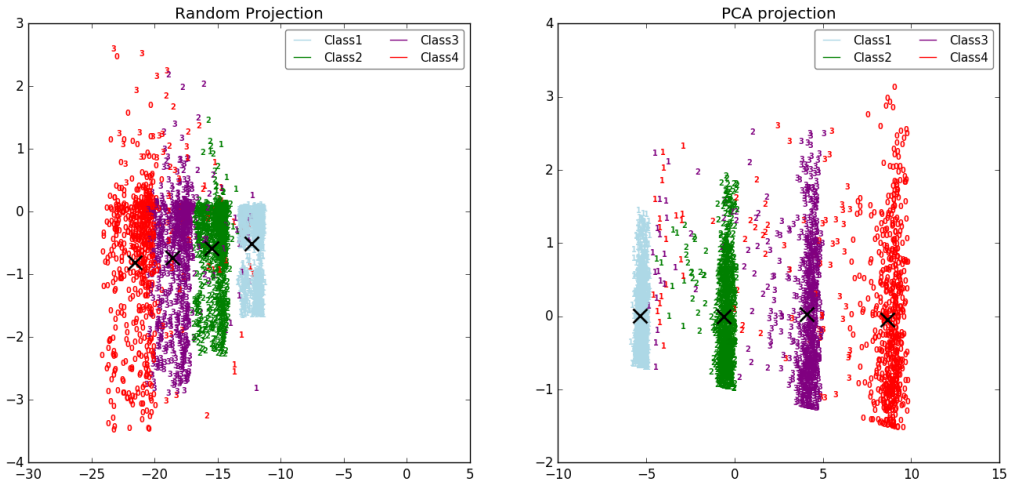
Figura 5.8: Riduzione dimensionale - Confronto fra feature vectors costituiti dalle coppie velocità e accelerazione nel caso di ordinamento decrescente



(a) Feature vectors: Coppie velocità e accelerazione - Lunghezza: 100



(b) Feature vectors: Coppie velocità e accelerazione - Lunghezza: 20



(c) Feature vectors: Coppie velocità e accelerazione - Lunghezza: 5

Capitolo 6

Risultati sperimentali scenario intermedio

6.1 I valori utilizzati

Nello scenario Increasing sf, come anticipato in precedenza, si sono considerati veicoli che differiscono solo per aggressività, ovvero per speed factor. In particolare, i valori scelti sono elencati in tabella 6.1.

Tabella 6.1: Valori utilizzati per le classi di veicoli nello scenario intermedio

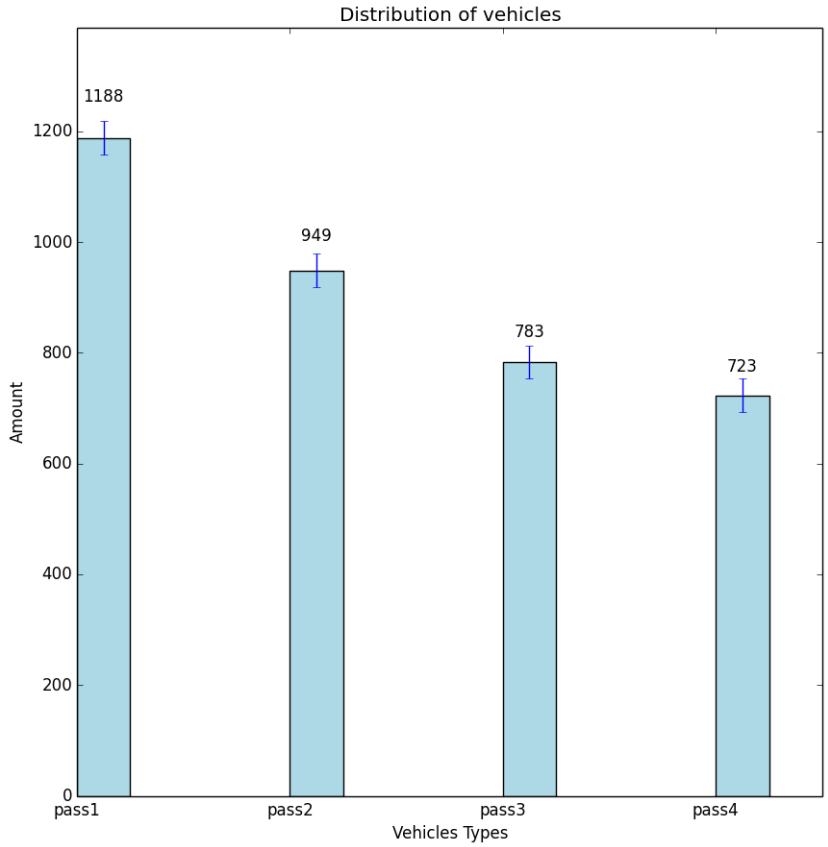
id	**accel**	**decel**	**speedFactor**
Class1	2.7	4.5	0.8
Class2	2.7	4.5	1.0
Class3	2.7	4.5	1.2
Class4	2.7	4.5	1.4

Come ci si poteva aspettare, in questo caso, k-means ha avuto maggiori difficoltà rispetto al caso precedente a eseguire il clustering in quanto tutte le tipologie avevano le medesime caratteristiche fisiche.

6.2 Analisi dei dati sequenziali

Per quel che concerne i dati sequenziali per lo scenario Increasing sf, l'applicazione del filtro a posteriori ha determinato una distribuzione disomogenea delle classi di veicoli con una prevalenza di Class1 (32.6%), seguiti nell'ordine da Class2 (26.1%), Class3 (21.5%) e Class4 (19.8%). Tale distribuzione è meglio rappresentata in figura 6.1.

Figura 6.1: Distribuzione delle classi di veicoli all'interno del sottoinsieme analizzato per lo scenario intermedio



Come per lo scenario precedente, nel seguito l'attenzione sarà data ai tre casi: feature vectors non ordinati, ordinati in senso crescente e in senso decrescente, per poi eseguire un confronto con il caso aggregato.

6.2.1 Feature vectors senza ordinamento

Per quel che concerne i feature vectors senza ordinamento, anche per lo scenario Increasing sf, il clustering è stato poco significativo. In particolare, anche questa volta si è osservato

un miglioramento delle metriche in modo inversamente proporzionale alla lunghezza dei features vector presi in esame, ma a differenza del caso precedente, si è anche visto di come sotto un certo valore limite le cose peggiorassero in modo molto evidente. (Tabella 6.2).

Tabella 6.2: Confronto delle metriche al variare della dimensione dei feature vectors (fv_length) - Caso non ordinati

Modality: Unsorted - Increasing sf
Used vehicles: 3643

mode	fv_length	homo	completeness	v_meas	ARI	AMI
S	100	0.006	0.006	0.006	0.004	0.005
S	50	0.207	0.207	0.207	0.188	0.206
S	25	0.44	0.492	0.464	0.493	0.439
S	10	0.641	0.678	0.659	0.677	0.641
S	5	0.001	0.001	0.001	0.0	0.0
A	91	0.177	0.176	0.176	0.184	0.175
A	50	0.085	0.084	0.084	0.071	0.083
A	25	0.26	0.258	0.259	0.258	0.257
A	10	0.573	0.571	0.572	0.6	0.57
A	5	0.001	0.001	0.001	-0.0	0.0
SA	100	0.207	0.207	0.207	0.188	0.206
SA	50	0.441	0.494	0.466	0.496	0.441
SA	25	0.716	0.739	0.727	0.755	0.715
SA	10	0.001	0.001	0.001	-0.0	0.0
SA	5	0.001	0.001	0.001	-0.0	0.0

6.2.2 Feature vectors con ordinamento crescente

Un discorso analogo a quanto visto per il caso All different può essere invece fatto per i dati ordinati in senso crescente. Anche in questo caso, infatti, si osserva di come i valori delle metriche siano bassi e di come non accennino a crescere con la riduzione della lunghezza dei features vectors. (Tabella 6.3).

6.2.3 Feature vectors con ordinamento decrescente

L'ordinamento in ordine decrescente si è rivelato anche in questo caso il migliore per il clustering. In particolare, però, come ci si poteva aspettare, il clustering sulle accelerazioni è

Tabella 6.3: Confronto delle metriche al variare della dimensione dei feature vectors (fv_length) - Caso ordinamento crescente

Modality: Sorted - Increasing sf
Used vehicles: 3643

mode	fv_length	homo	completeness	v_meas	ARI	AMI
S Sorted	100	0.03	0.034	0.032	0.001	0.029
S Sorted	50	0.022	0.052	0.031	-0.013	0.021
S Sorted	25	0.012	0.075	0.02	-0.007	0.011
S Sorted	10	0.004	0.142	0.008	-0.002	0.003
S Sorted	5	0.002	0.12	0.004	-0.001	0.001
A Sorted	91	0.13	0.14	0.135	0.105	0.13
A Sorted	50	0.081	0.089	0.085	0.071	0.081
A Sorted	25	0.04	0.044	0.042	0.024	0.039
A Sorted	10	0.022	0.032	0.026	0.002	0.021
A Sorted	5	0.013	0.025	0.017	-0.003	0.012
SA Sorted	100	0.011	0.012	0.011	-0.003	0.01
SA Sorted	50	0.01	0.016	0.013	-0.01	0.009
SA Sorted	25	0.027	0.034	0.03	0.028	0.026
SA Sorted	10	0.022	0.026	0.024	0.027	0.021
SA Sorted	5	0.019	0.031	0.023	0.019	0.018

stato del tutto inefficace in questo caso, mentre hanno avuto maggiore successo quello sulle velocità e quello sulle coppie di velocità e accelerazione (Tabella 6.4).

6.3 Confronto con i dati aggregati

Come ci si poteva aspettare, il clustering effettuato sulla base dei dati aggregati e risultato il più efficace ponendosi come limite superiore per tutti gli altri approcci. In tabella 6.5 sono mostrati i migliori risultati del clustering sui dati sequenziali a confronto con quelli ottenuti nel caso aggregato. Analogamente allo scenario All different, i risultati del caso con ordinamento crescente sono stati omessi in quanto non considerati rilevanti. In figura 6.2 sono invece mostrati i grafici relativi. Seguono una serie di grafici ottenuti attraverso la riduzione dimensionale che vanno a mostrare in modo evidente l'andamento dei dati sequenziali al variare delle dimensioni dei feature vectors usati nei casi senza ordinamento e con ordinamento decrescente.

Tabella 6.4: Confronto delle metriche al variare della dimensione dei feature vectors (fv_length) - Caso ordinamento decrescente

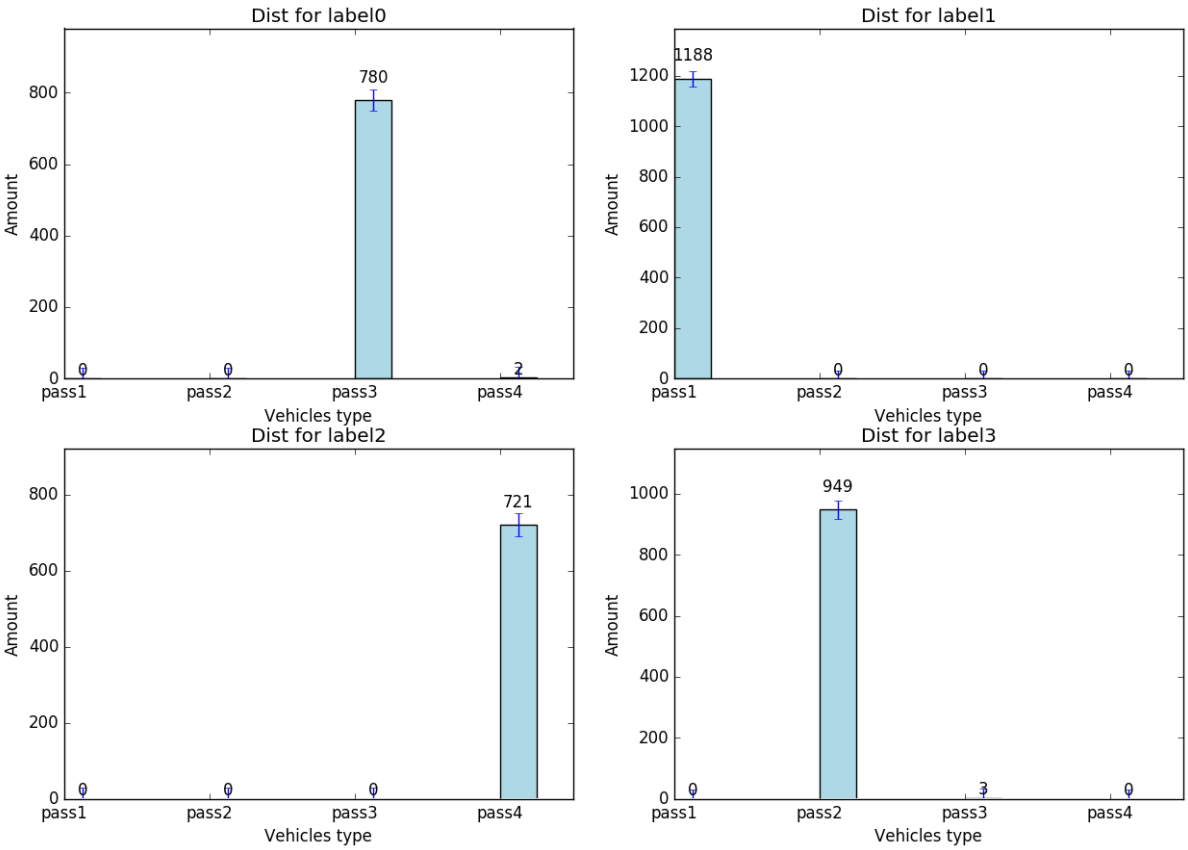
Modality: Reversed - Increasing sf
Used vehicles: 3643

mode	fv_length	homo	completeness	v_meas	ARI	AMI
S Reversed	100	0.706	0.715	0.71	0.734	0.706
S Reversed	50	0.853	0.862	0.857	0.877	0.852
S Reversed	25	0.943	0.945	0.944	0.963	0.943
S Reversed	10	0.978	0.978	0.978	0.988	0.978
S Reversed	5	0.987	0.988	0.988	0.994	0.987
A Reversed	91	0.296	0.296	0.296	0.243	0.295
A Reversed	50	0.318	0.326	0.322	0.233	0.317
A Reversed	25	0.246	0.278	0.261	0.177	0.245
A Reversed	10	0.138	0.184	0.158	0.103	0.137
A Reversed	5	0.09	0.121	0.103	0.06	0.089
SA Reversed	100	0.565	0.582	0.573	0.615	0.564
SA Reversed	50	0.671	0.703	0.687	0.711	0.671
SA Reversed	25	0.771	0.787	0.779	0.813	0.771
SA Reversed	10	0.854	0.862	0.858	0.889	0.853
SA Reversed	5	0.884	0.889	0.887	0.914	0.884

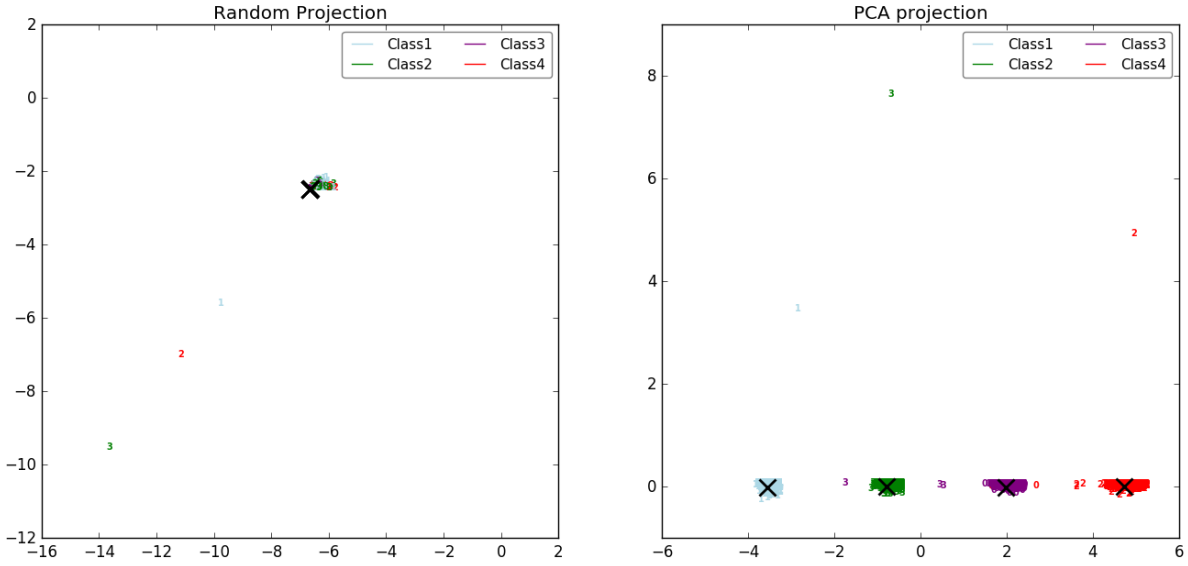
Tabella 6.5: Confronto fra i migliori risultati ottenuti nello scenario intermedio: h sta per omogeneità, c per completezza, v per V-measure. Per le tracce S sta per speed, A per accelerazione e SA per la combinazione di queste due. Agg indica i risultati per le feature aggregate

Trace	Sorting	m	h	c	v	ARI	AMI
S	None	15	0.656	0.685	0.670	0.697	0.656
A	None	15	0.572	0.570	0.571	0.602	0.569
SA	None	25	0.716	0.739	0.727	0.755	0.715
S	Dec	5	0.987	0.988	0.988	0.994	0.987
A	Dec	50	0.318	0.326	0.322	0.233	0.317
SA	Dec	5	0.884	0.889	0.887	0.914	0.884
Agg	–	–	0.993	0.993	0.993	0.997	0.993

Figura 6.2: Grafici relativi alla distribuzione dei veicoli nel caso di dati aggregati

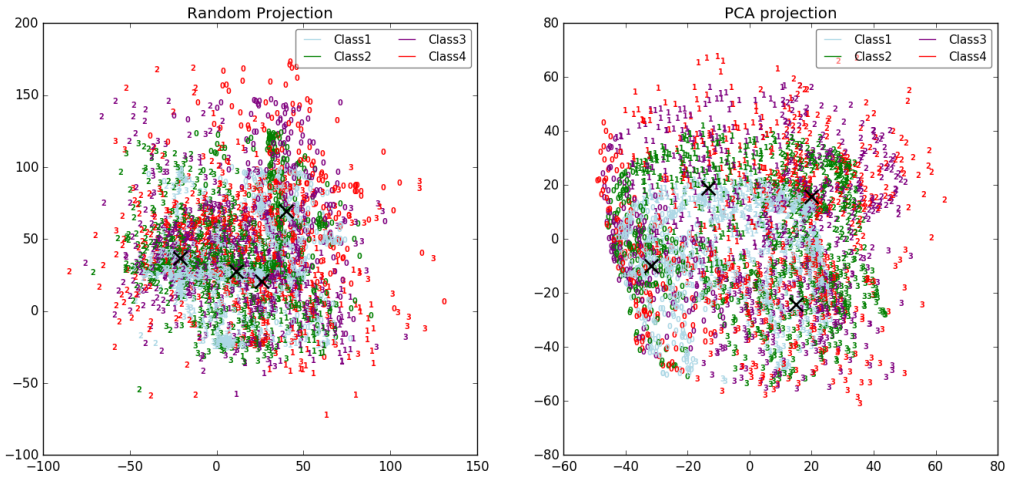


(a) Distribuzione delle label k-means nel caso aggregato

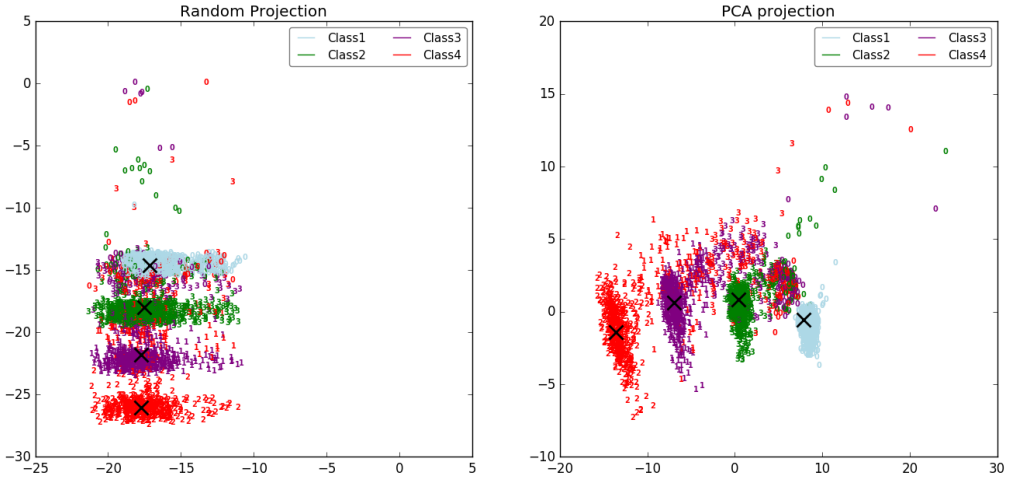


(b) Riduzione dimensionale nel caso aggregato

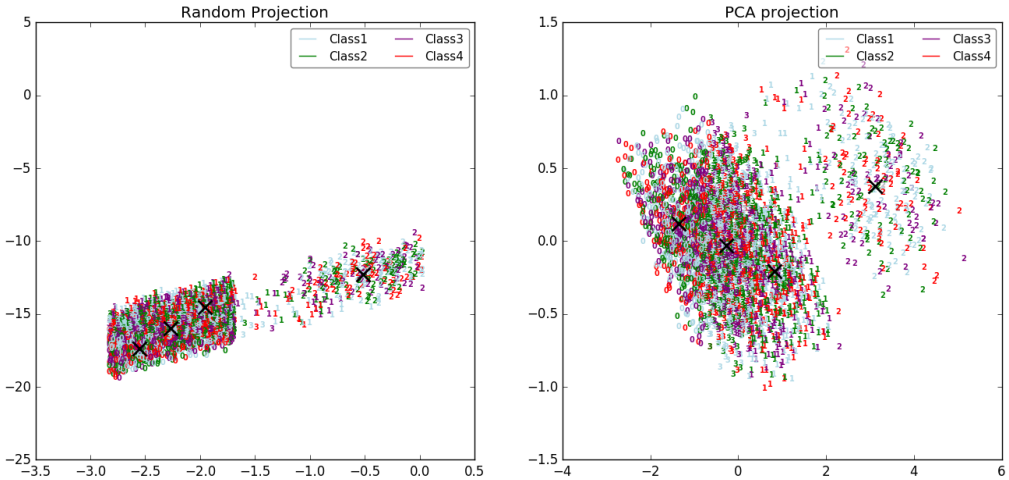
Figura 6.3: Riduzione dimensionale - Confronto fra feature vectors costituiti dalle sole velocità nel caso senza ordinamento



(a) Feature vectors: Serie delle velocità - Lunghezza: 100

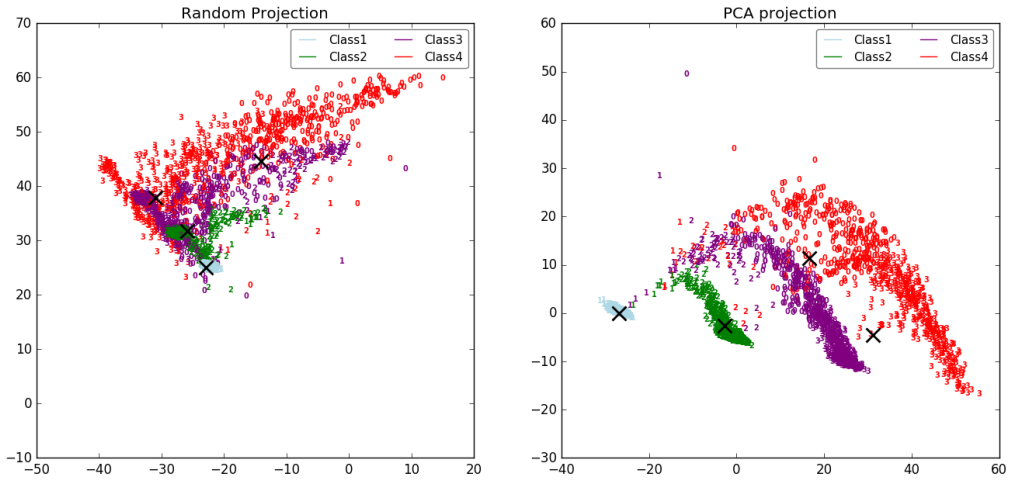


(b) Feature vectors: Serie delle velocità - Lunghezza: 15

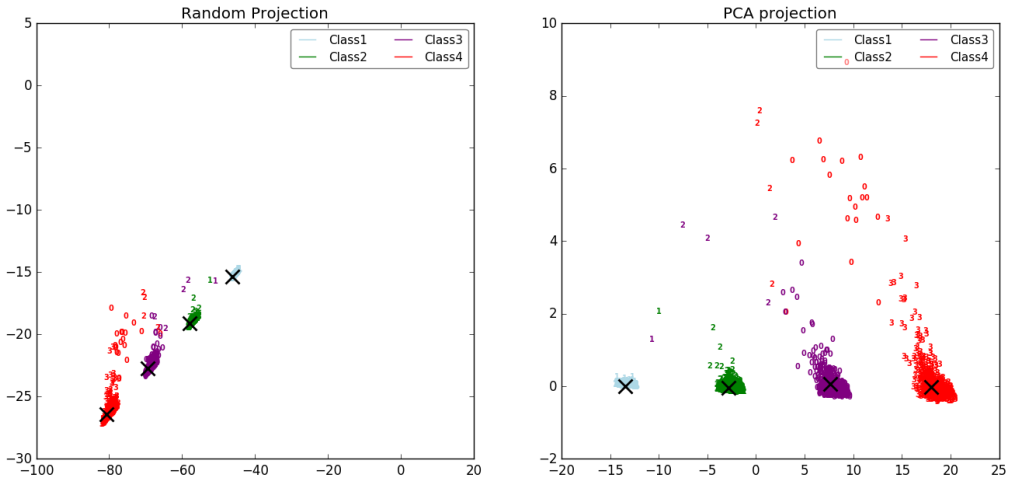


(c) Feature vectors: Serie delle velocità - Lunghezza: 5

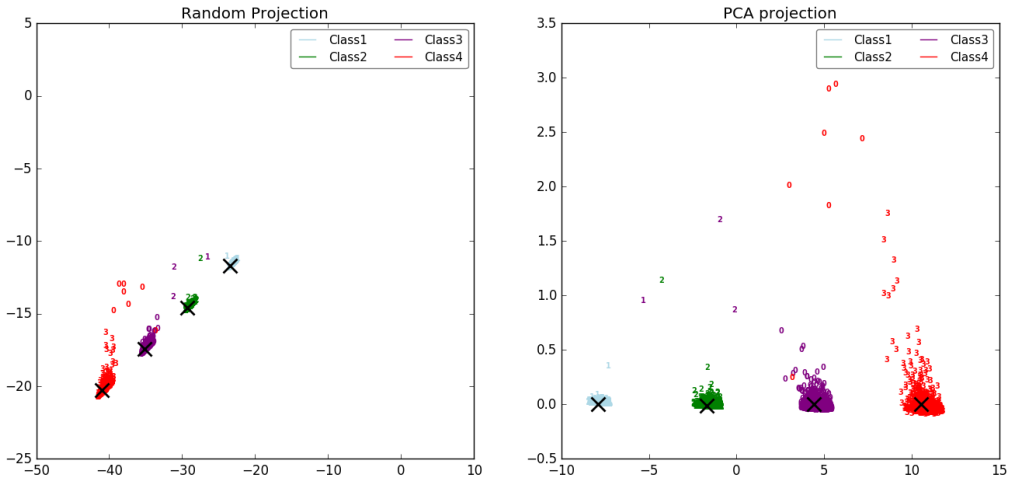
Figura 6.4: Riduzione dimensionale - Confronto fra feature vectors costituiti dalle sole velocità nel caso di ordinamento decrescente



(a) Feature vectors: Serie delle velocità - Lunghezza: 100

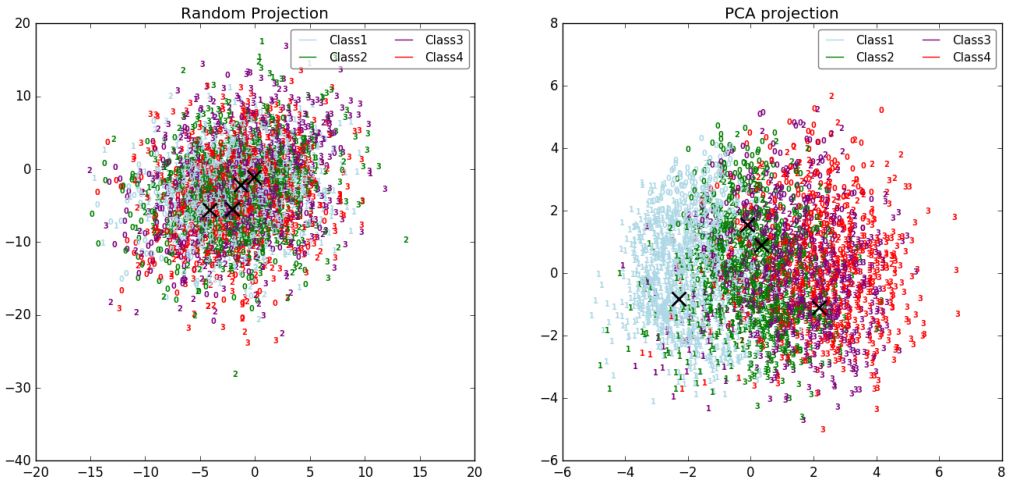


(b) Feature vectors: Serie delle velocità - Lunghezza: 15

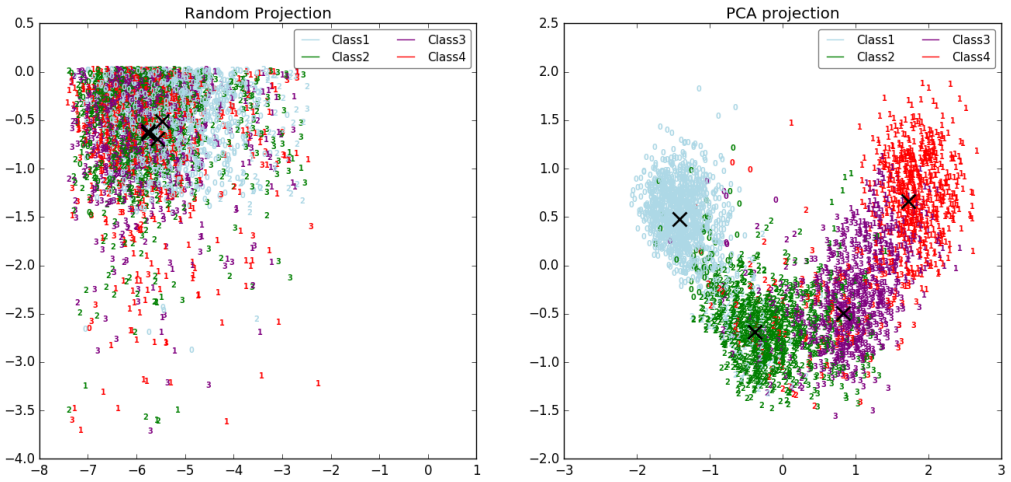


(c) Feature vectors: Serie delle velocità - Lunghezza: 5

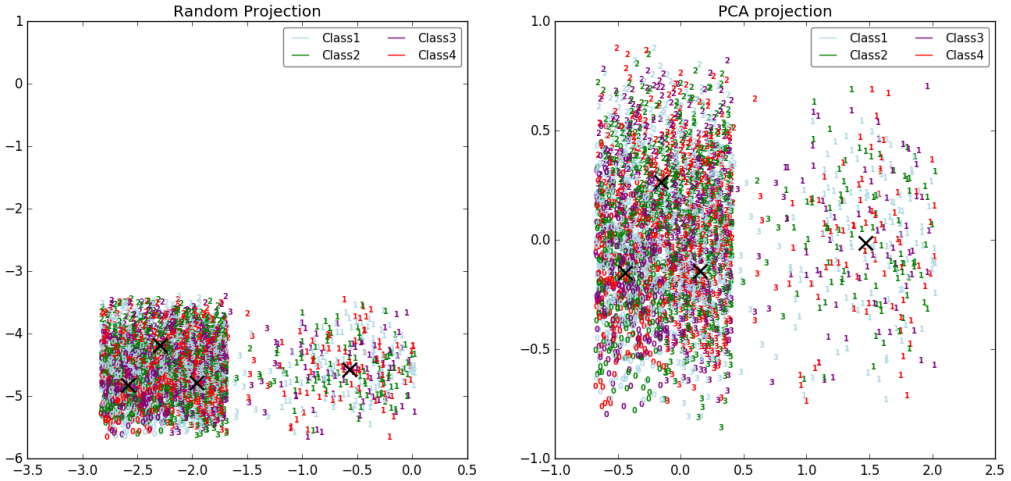
Figura 6.5: Riduzione dimensionale - Confronto fra feature vectors costituiti dalle sole accelerazioni nel caso senza ordinamento



(a) Feature vectors: Serie delle accelerazioni - Lunghezza: 91

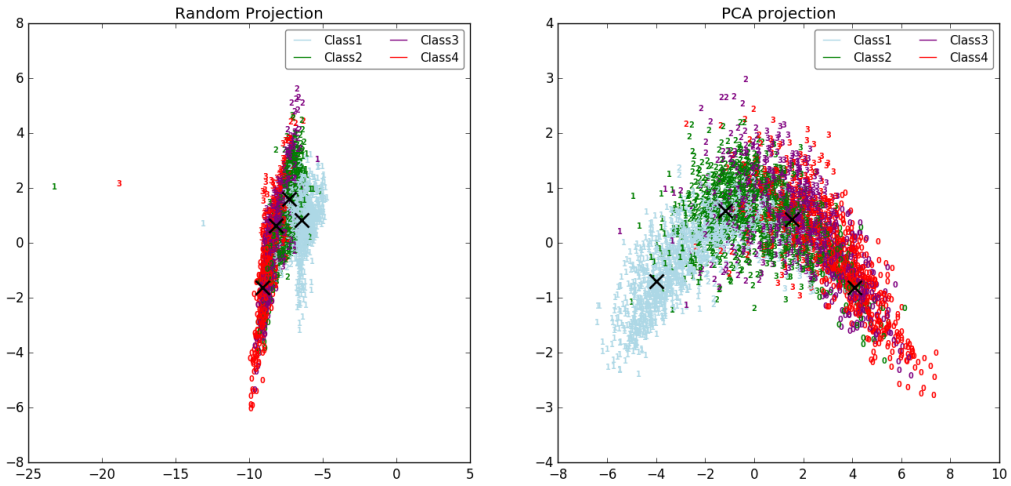


(b) Feature vectors: Serie delle accelerazioni - Lunghezza: 15



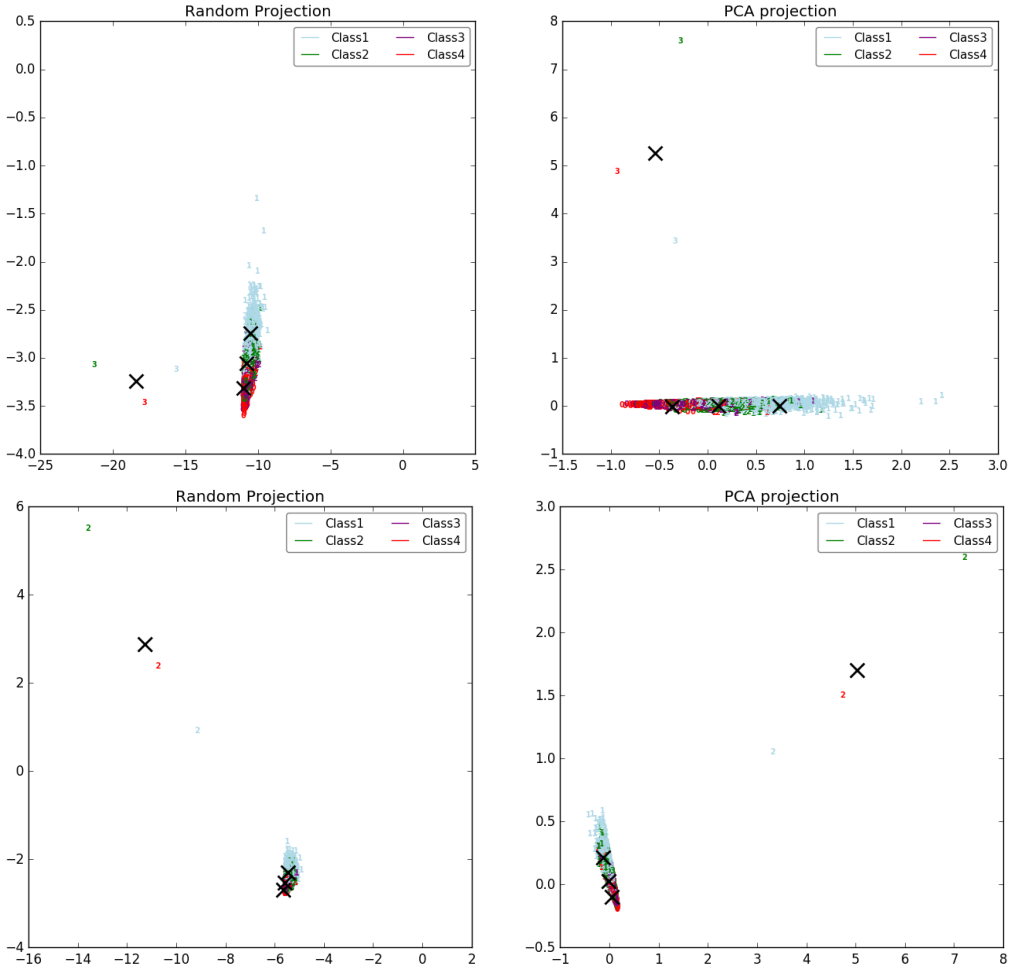
(c) Feature vectors: Serie delle accelerazioni - Lunghezza: 5

Figura 6.6: Riduzione dimensionale - Confronto fra feature vectors costituiti dalle sole accelerazioni nel caso di ordinamento decrescente



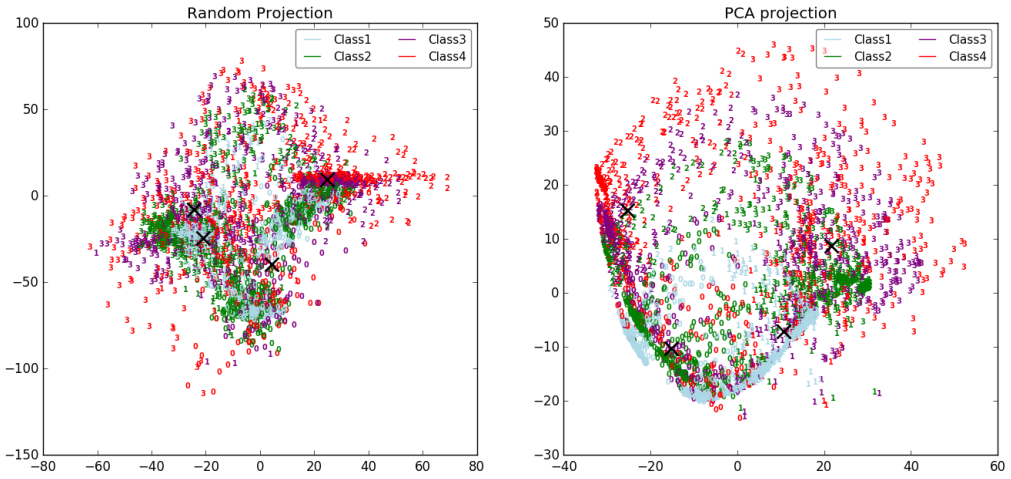
(a) Feature vectors: Serie delle accelerazioni - Lunghezza: 91

(b) Feature vectors: Serie delle accelerazioni - Lunghezza: 15

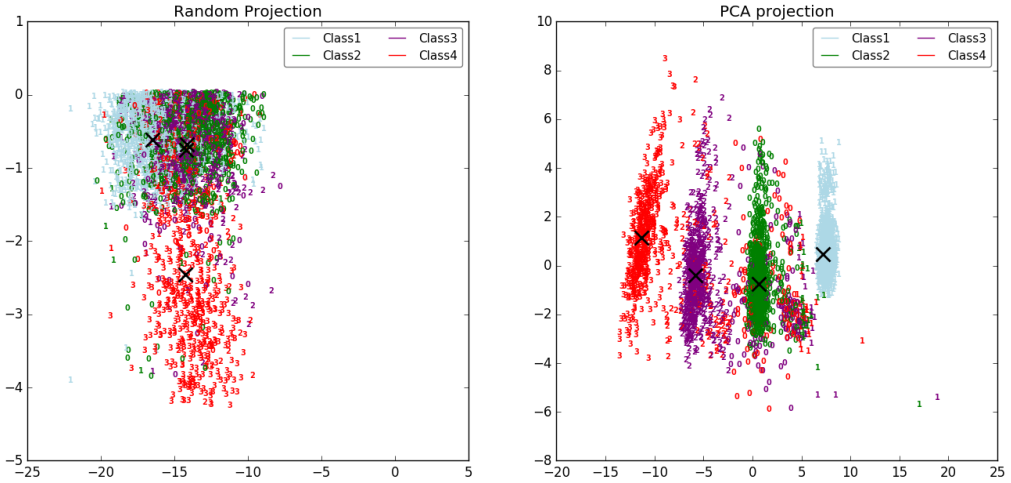


(c) Feature vectors: Serie delle accelerazioni - Lunghezza: 5

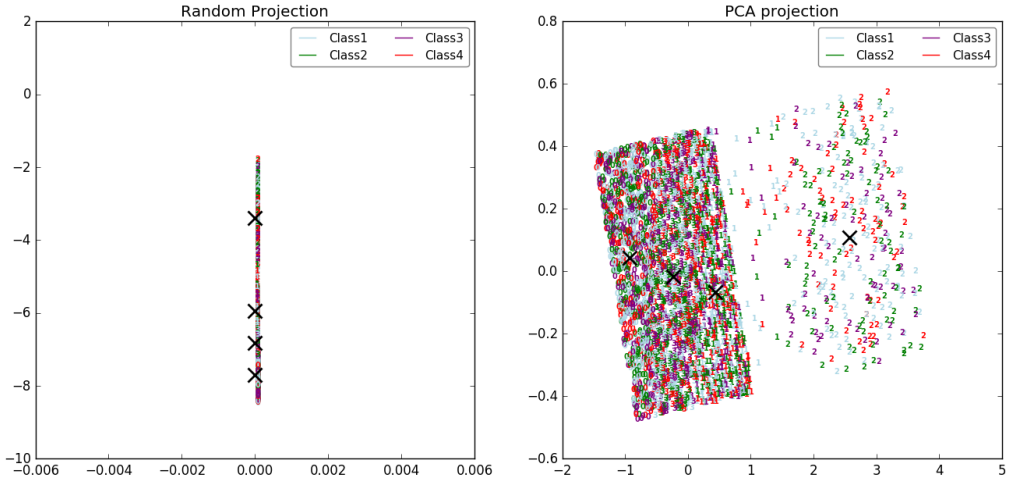
Figura 6.7: Riduzione dimensionale - Confronto fra feature vectors costituiti dalle coppie velocità e accelerazione nel caso senza ordinamento



(a) Feature vectors: Coppie velocità e accelerazione - Lunghezza: 100

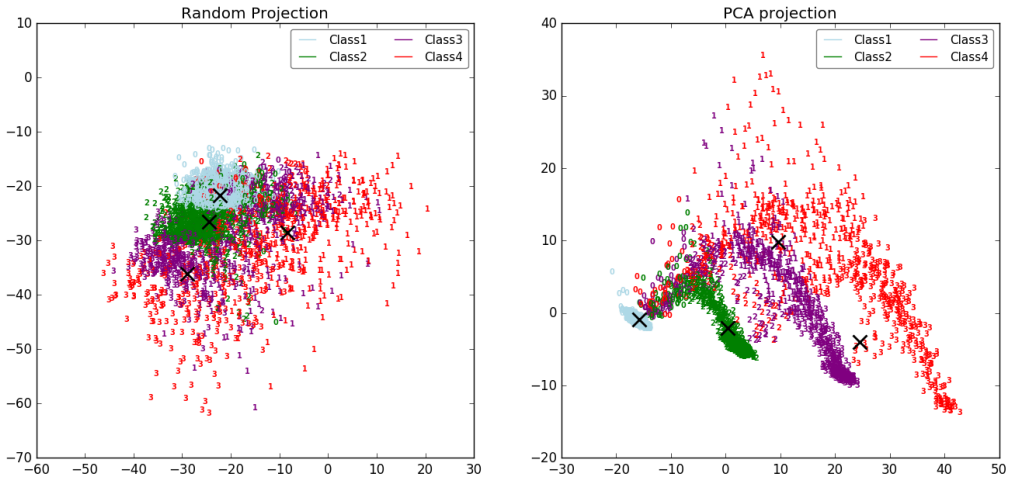


(b) Feature vectors: Coppie velocità e accelerazione - Lunghezza: 25

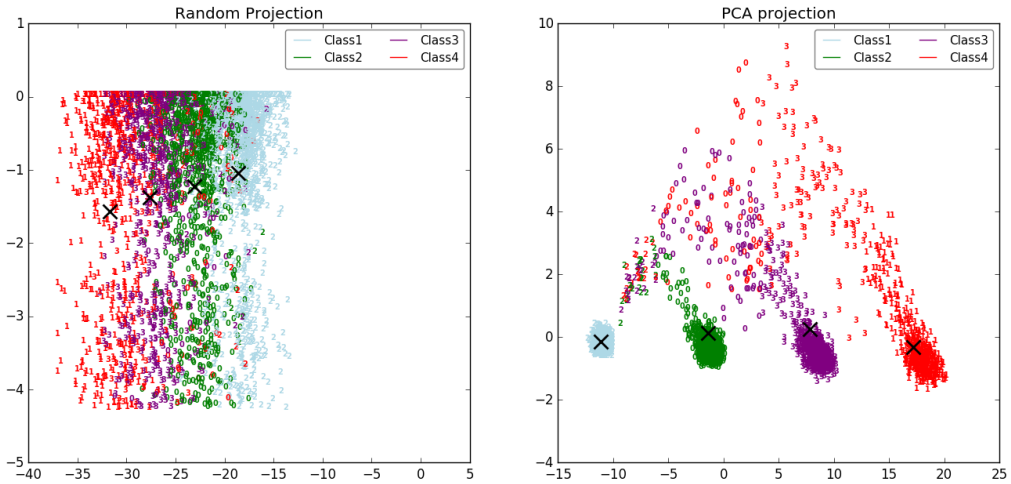


(c) Feature vectors: Coppie velocità e accelerazione - Lunghezza: 5

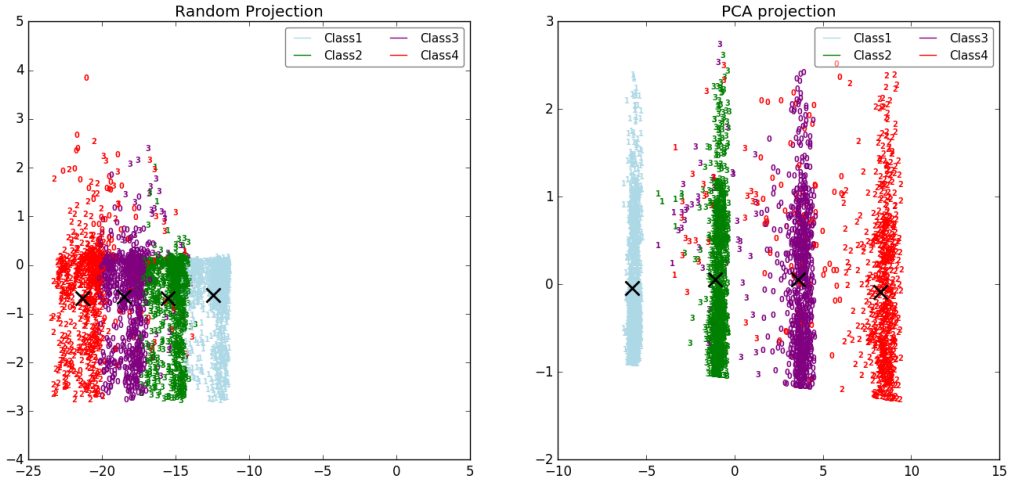
Figura 6.8: Riduzione dimensionale - Confronto fra feature vectors costituiti dalle coppie velocità e accelerazione nel caso di ordinamento decrescente



(a) Feature vectors: Coppie velocità e accelerazione - Lunghezza: 100



(b) Feature vectors: Coppie velocità e accelerazione - Lunghezza: 25



(c) Feature vectors: Coppie velocità e accelerazione - Lunghezza: 5

Capitolo 7

Risultati sperimentali nello scenario difficile

7.1 I valori utilizzati

Lo scenario Merged, come menzionato in precedenza, considera veicoli che hanno le stesse caratteristiche fisiche, ma con una differente aggressività. In particolare, i valori scelti sono elencati in tabella 7.1.

Tabella 7.1: Valori utilizzati per le classi di veicoli nello scenario difficile

id	**accel**	**decel**	**speedFactor**
Class1	2.2	4.5	0.8
Class2	2.8	4.5	0.8
Class3	2.2	4.5	1.2
Class4	2.8	4.5	1.2

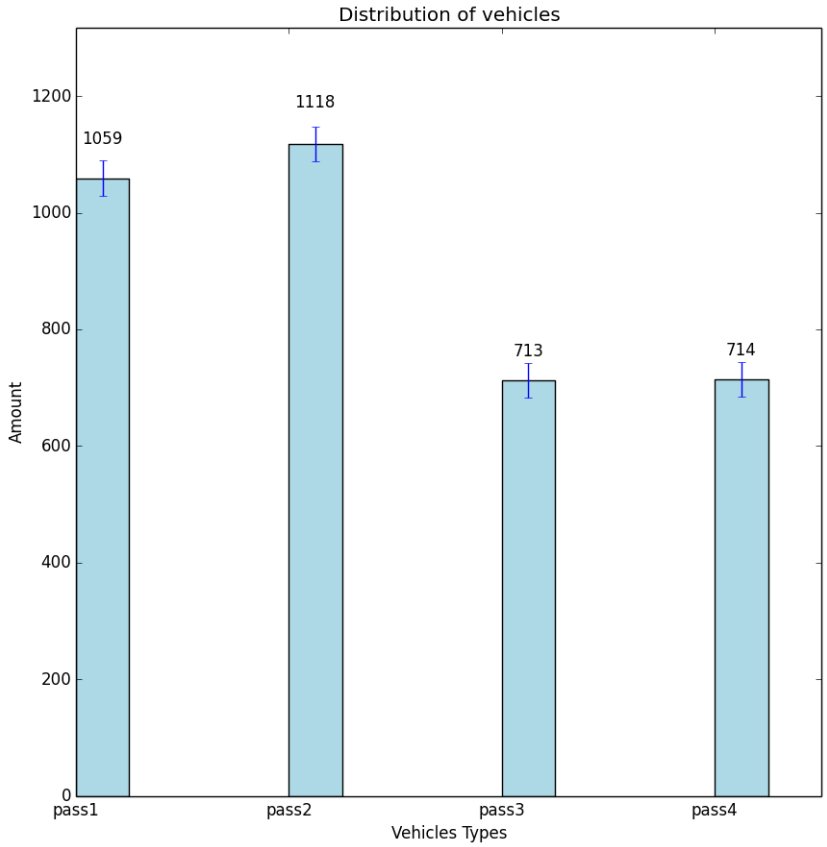
Tale scenario si è configurato come quello più complicato da riconoscere in quanto k-means tendeva a raggruppare a coppie le quattro tipologie di veicoli.

7.2 Analisi dei dati sequenziali

Per quel che concerne i dati sequenziali per lo scenario Merged, l'applicazione del filtro a posteriori ha determinato una distribuzione disomogenea delle classi di veicoli con una

prevalenza di Class2 (31.0%), seguiti nell'ordine da Class1 (29.4%), Class3 (19.8%) e Class4 (19.8%). Tale distribuzione è meglio rappresentata in figura 7.1.

Figura 7.1: Distribuzione delle classi di veicoli all'interno del sottoinsieme analizzato per lo scenario difficile



Come per i precedenti scenari, nel seguito l'attenzione sarà data ai tre casi feature vectors non ordinati, ordinati in senso crescente e in senso decrescente, per poi eseguire un confronto con il caso aggregato.

7.2.1 Feature vectors senza ordinamento

Per quel che concerne i feature vectors senza ordinamento, per lo scenario Merged le cose sono andate in modo diverso rispetto ai due scenari predecenti. In particolare, il clustering è stato maggiormente significativo nel caso di features vectors non ordinati rispetto a features vector ordinati o ordinati in senso decrescente. Anche in questo caso, comunque, si è visto un miglioramento dei risultati mano a mano che si accorciavano i features vectors utilizzati (Tabella 7.2).

Tabella 7.2: Confronto delle metriche al variare della dimensione dei feature vectors (fv_length) - Caso non ordinati

Modality: Unsorted - Merged
Used vehicles: 3604

mode	fv_length	homo	completeness	v_meas	ARI	AMI
S	100	0.001	0.001	0.001	-0.001	-0.0
S	50	0.127	0.132	0.129	0.092	0.126
S	25	0.359	0.499	0.418	0.332	0.359
S	10	0.723	0.737	0.73	0.741	0.723
S	5	0.301	0.302	0.302	0.236	0.301
A	100	0.245	0.242	0.243	0.19	0.241
A	50	0.115	0.114	0.115	0.098	0.113
A	25	0.335	0.353	0.344	0.285	0.335
A	10	0.702	0.714	0.708	0.738	0.702
A	5	0.334	0.38	0.356	0.317	0.333
SA	100	0.127	0.132	0.13	0.092	0.126
SA	50	0.369	0.507	0.427	0.341	0.368
SA	25	0.657	0.682	0.669	0.676	0.657
SA	10	0.302	0.332	0.316	0.266	0.302
SA	5	0.217	0.239	0.227	0.167	0.216

7.2.2 Feature vectors con ordinamento crescente

Come ci si poteva aspettare, il clustering sulle serie ordinate non ha dato alcun risultato anche in questo caso attestandosi su valori molto bassi (Tabella: 7.3).

7.2.3 Feature vectors con ordinamento decrescente

Contrariamente alle aspettative, in questo scenario, l'ordinamento in senso decrescente dei features vectors, pur essendo migliore della sua controparte, non ha dato buoni risultati impedendo un clustering chiaro (Tabelle 7.4).

7.3 Confronto con i dati aggregati

Come da aspettative, anche in questo scenario il clustering effettuato sulla base dei dati aggregati e risultato il più efficace ponendosi come limite superiore per tutti gli altri

Tabella 7.3: Confronto delle metriche al variare della dimensione dei feature vectors (fv_length) - Caso ordinamento crescente

Modality: Sorted - Merged
Used vehicles: 3604

mode	fv_length	homo	completeness	v_meas	ARI	AMI
S Sorted	100	0.031	0.034	0.032	0.001	0.03
S Sorted	50	0.02	0.04	0.027	-0.013	0.019
S Sorted	25	0.009	0.052	0.016	-0.006	0.008
S Sorted	10	0.004	0.072	0.007	-0.002	0.003
S Sorted	5	0.002	0.105	0.004	-0.001	0.001
A Sorted	100	0.204	0.216	0.21	0.172	0.203
A Sorted	50	0.109	0.116	0.112	0.085	0.108
A Sorted	25	0.052	0.057	0.055	0.03	0.052
A Sorted	10	0.027	0.039	0.032	0.007	0.026
A Sorted	5	0.013	0.023	0.017	-0.0	0.012
SA Sorted	100	0.012	0.013	0.012	-0.001	0.011
SA Sorted	50	0.006	0.009	0.007	-0.009	0.005
SA Sorted	25	0.005	0.01	0.006	-0.009	0.004
SA Sorted	10	0.026	0.038	0.031	0.034	0.025
SA Sorted	5	0.024	0.038	0.03	0.03	0.023

approcci. In tabella 7.5 sono mostrati i migliori risultati del clustering sui dati sequenziali a confronto con quelli ottenuti nel caso aggregato. Analogamente agli scenari All different e Increasing-sf i risultati del caso con ordinamento crescente sono stati omessi in quanto non considerati rilevanti. In figura 7.2 sono invece mostrati i grafici relativi. Seguono una serie di grafici ottenuti attraverso la riduzione dimensionale che vanno a mostrare in modo evidente l'andamento dei dati sequenziali al variare delle dimensioni dei feature vectors usati nei casi senza ordinamento e con ordinamento decrescente.

Tabella 7.4: Confronto delle metriche al variare della dimensione dei feature vectors (fv_length) - Caso ordinamento decrescente

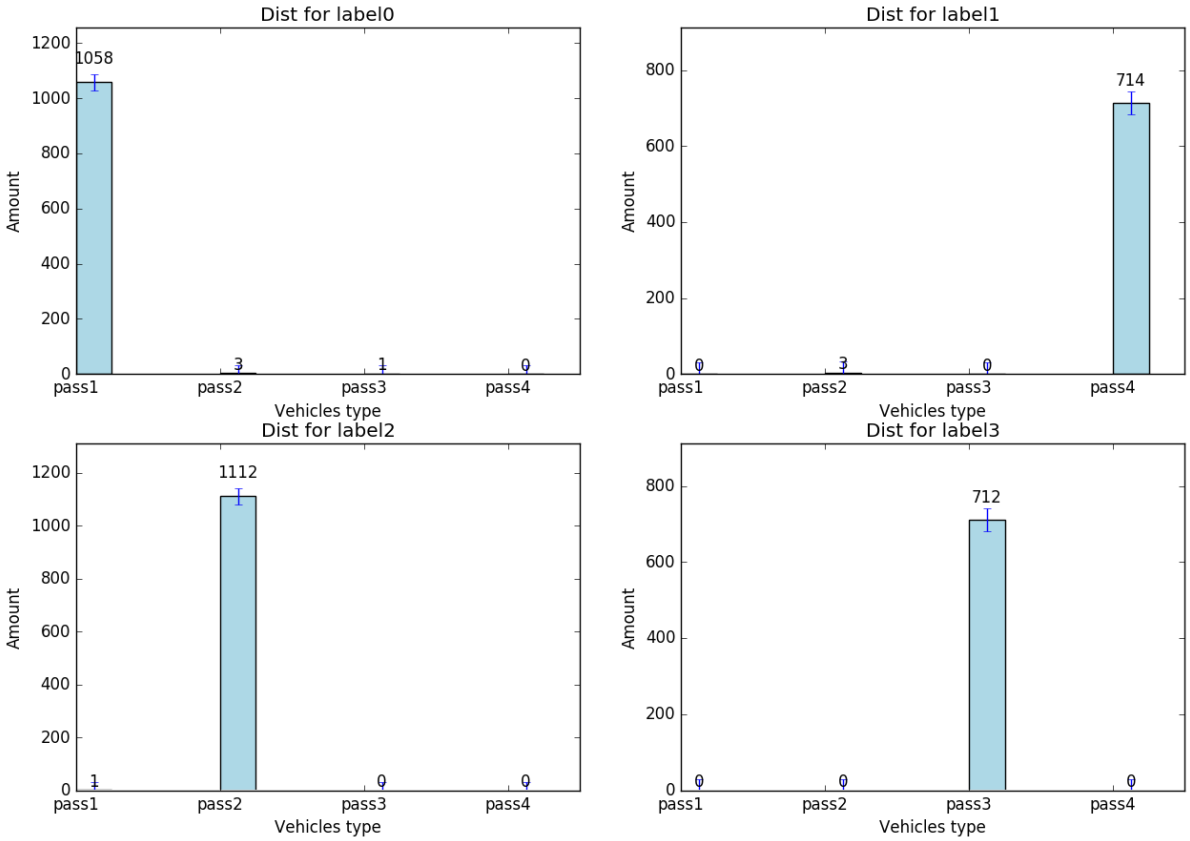
Modality: Reversed - Merged
Used vehicles: 3604

mode	fv_length	homo	completeness	v_meas	ARI	AMI
S Reversed	100	0.474	0.596	0.528	0.427	0.474
S Reversed	50	0.481	0.725	0.579	0.46	0.481
S Reversed	25	0.485	0.683	0.568	0.446	0.485
S Reversed	10	0.489	0.697	0.575	0.447	0.488
S Reversed	5	0.489	0.49	0.49	0.323	0.489
A Reversed	100	0.443	0.456	0.45	0.337	0.443
A Reversed	50	0.49	0.485	0.487	0.383	0.484
A Reversed	25	0.59	0.661	0.623	0.509	0.59
A Reversed	10	0.583	0.584	0.584	0.457	0.583
A Reversed	5	0.529	0.711	0.607	0.464	0.529
SA Reversed	100	0.392	0.511	0.444	0.364	0.391
SA Reversed	50	0.422	0.573	0.486	0.399	0.422
SA Reversed	25	0.46	0.695	0.554	0.446	0.46
SA Reversed	10	0.465	0.579	0.516	0.399	0.465
SA Reversed	5	0.473	0.596	0.528	0.406	0.473

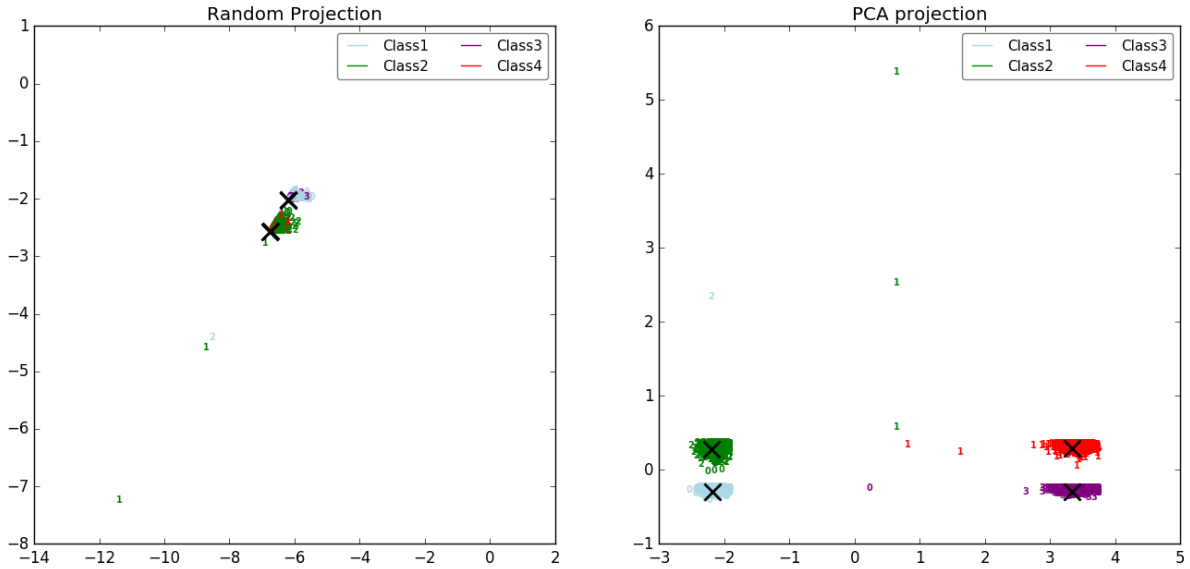
Tabella 7.5: Confronto fra i migliori risultati ottenuti nello scenario difficile: h sta per omogeneità, c per completezza, v per V-measure. Per le tracce S sta per speed, A per accelerazione e SA per la combinazione di queste due. Agg indica i risultati per le feature aggregate

Trace	Sorting	m	h	c	v	ARI	AMI
S	None	10	0.723	0.737	0.730	0.741	0.723
A	None	15	0.712	0.723	0.718	0.748	0.712
SA	None	20	0.727	0.740	0.733	0.747	0.727
S	Dec	30	0.485	0.843	0.616	0.476	0.484
A	Dec	25	0.590	0.661	0.623	0.509	0.590
SA	Dec	15	0.474	0.740	0.578	0.459	0.473
Agg	–	–	0.989	0.988	0.989	0.994	0.988

Figura 7.2: Grafici relativi alla distribuzione dei veicoli nel caso di dati aggregati

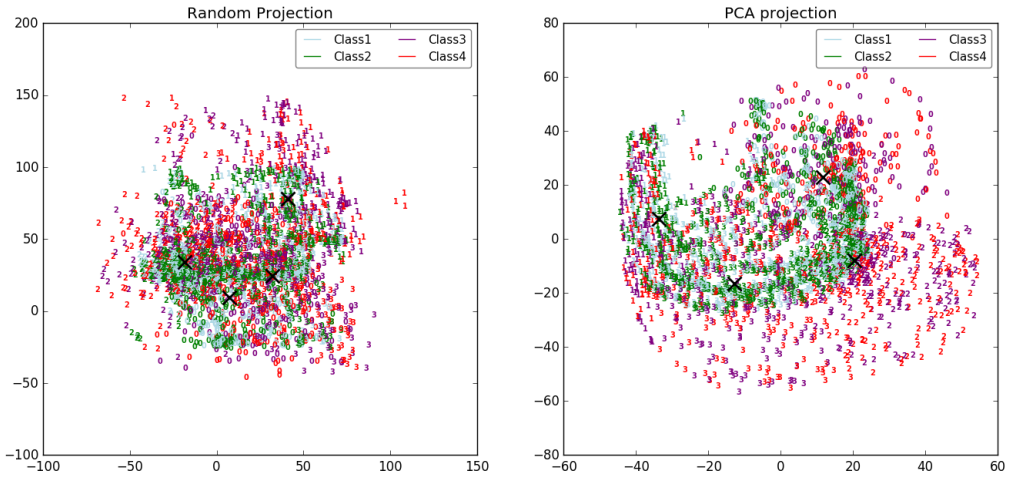


(a) Distribuzione delle label k-means nel caso aggregato

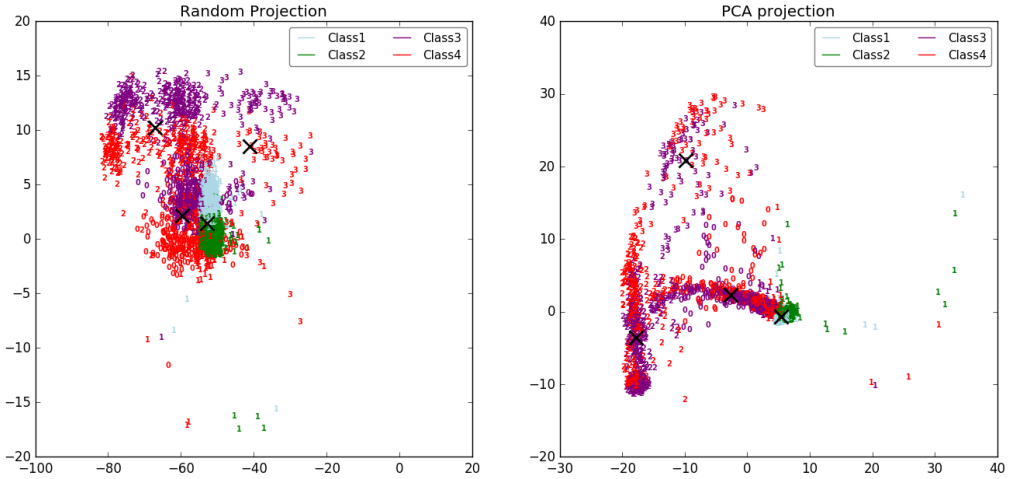


(b) Riduzione dimensionale nel caso aggregato

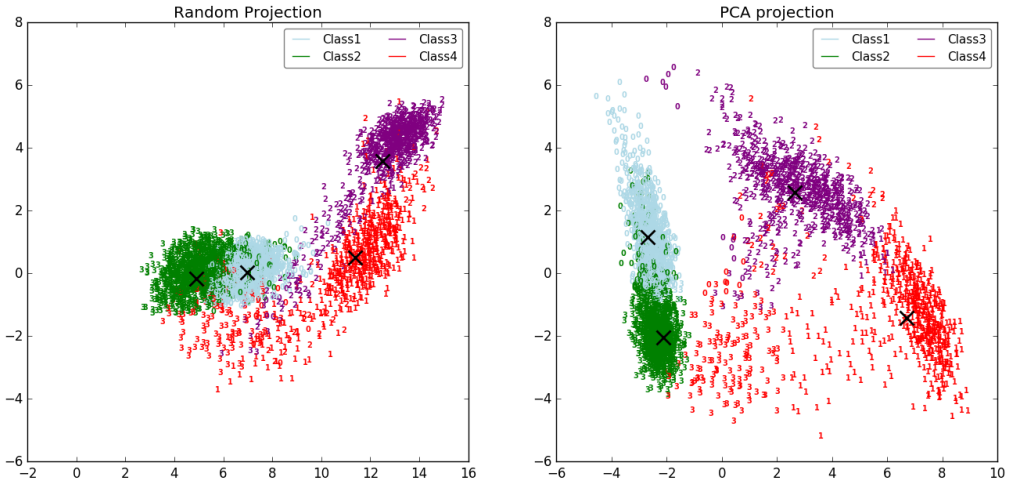
Figura 7.3: Riduzione dimensionale - Confronto fra feature vectors costituiti dalle sole velocità nel caso senza ordinamento



(a) Feature vectors: Serie delle velocità - Lunghezza: 100

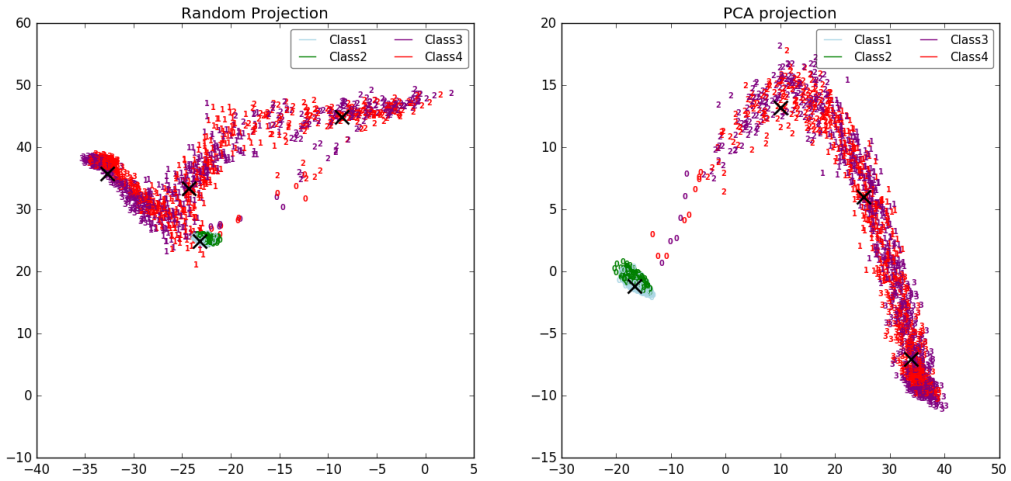


(b) Feature vectors: Serie delle velocità - Lunghezza: 30

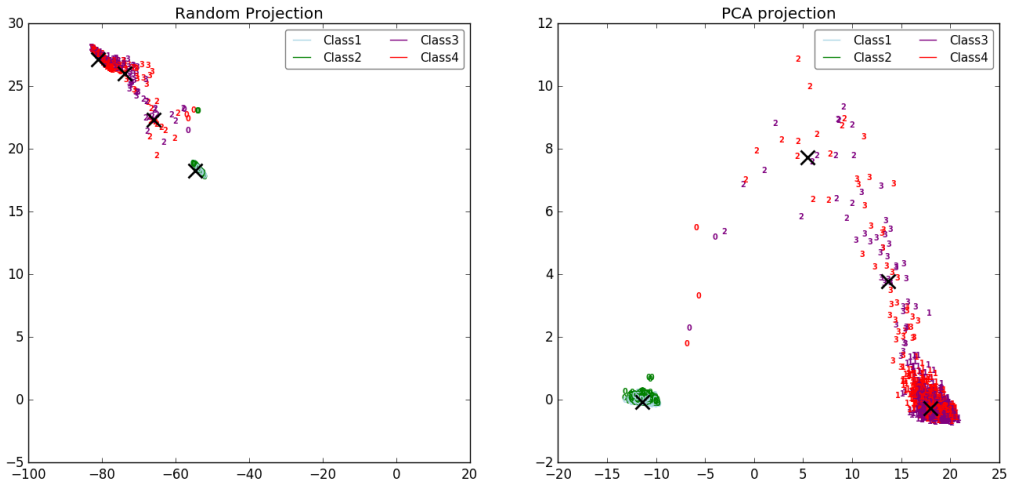


(c) Feature vectors: Serie delle velocità - Lunghezza: 10

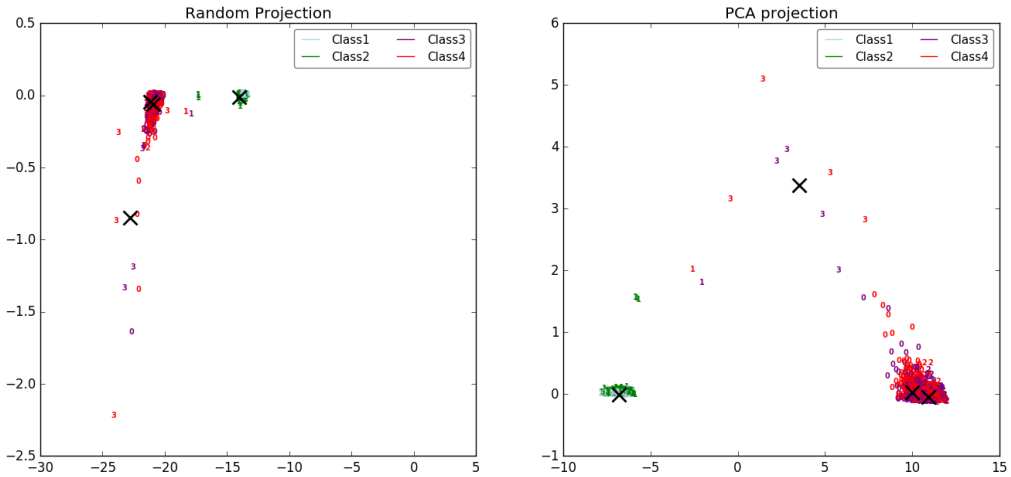
Figura 7.4: Riduzione dimensionale - Confronto fra feature vectors costituiti dalle sole velocità nel caso di ordinamento decrescente



(a) Feature vectors: Serie delle velocità - Lunghezza: 100

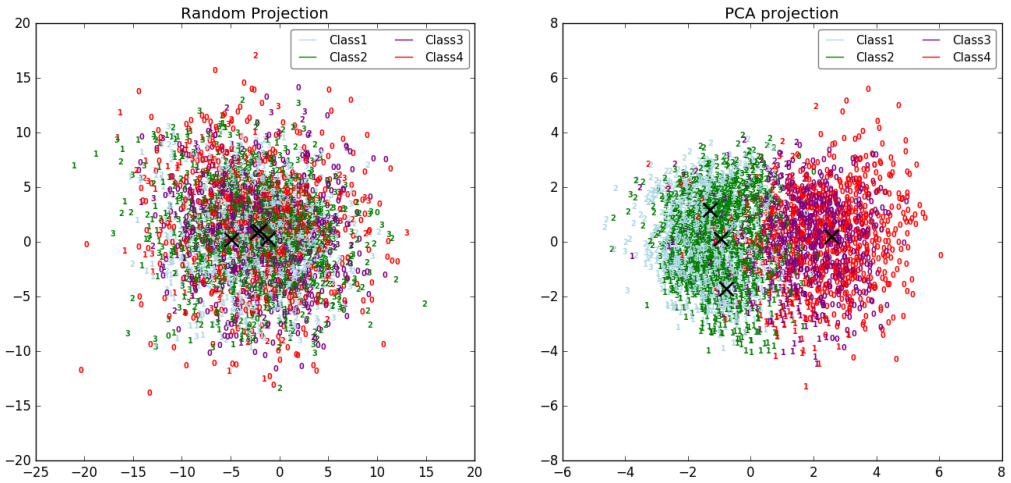


(b) Feature vectors: Serie delle velocità - Lunghezza: 30

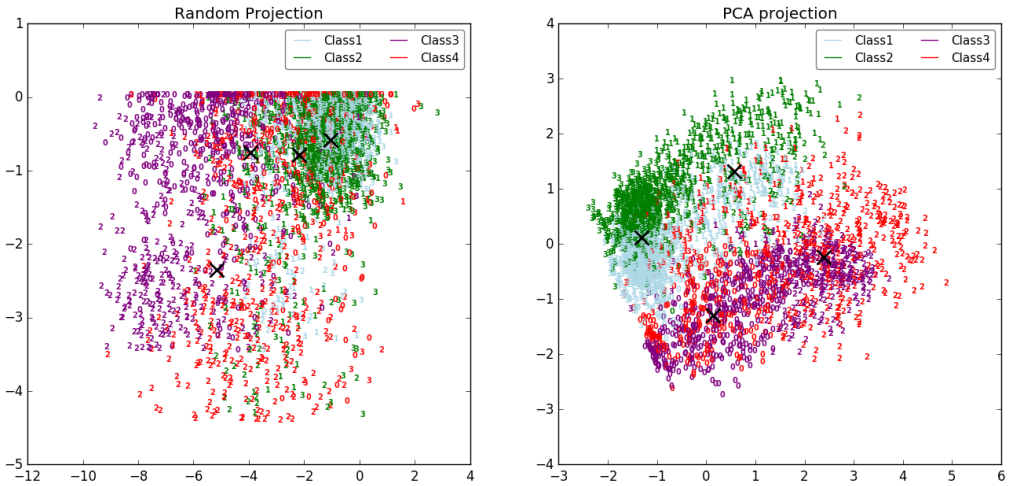


(c) Feature vectors: Serie delle velocità - Lunghezza: 10

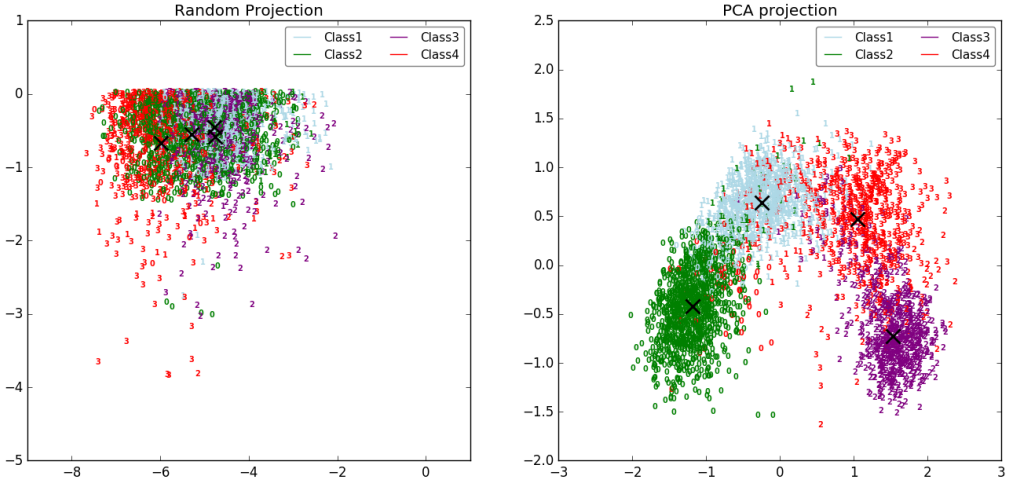
Figura 7.5: Riduzione dimensionale - Confronto fra feature vectors costituiti dalle sole accelerazioni nel caso senza ordinamento



(a) Feature vectors: Serie delle accelerazioni - Lunghezza: 100

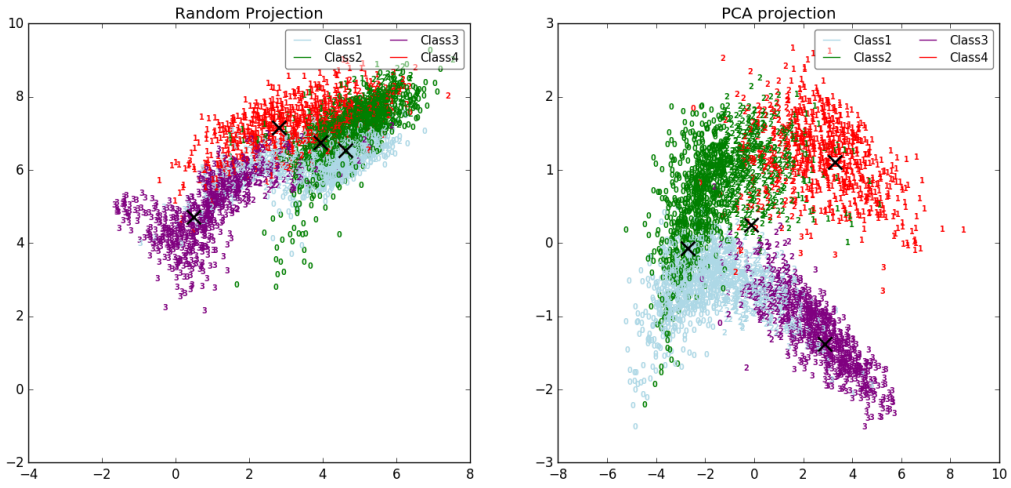


(b) Feature vectors: Serie delle accelerazioni - Lunghezza: 25

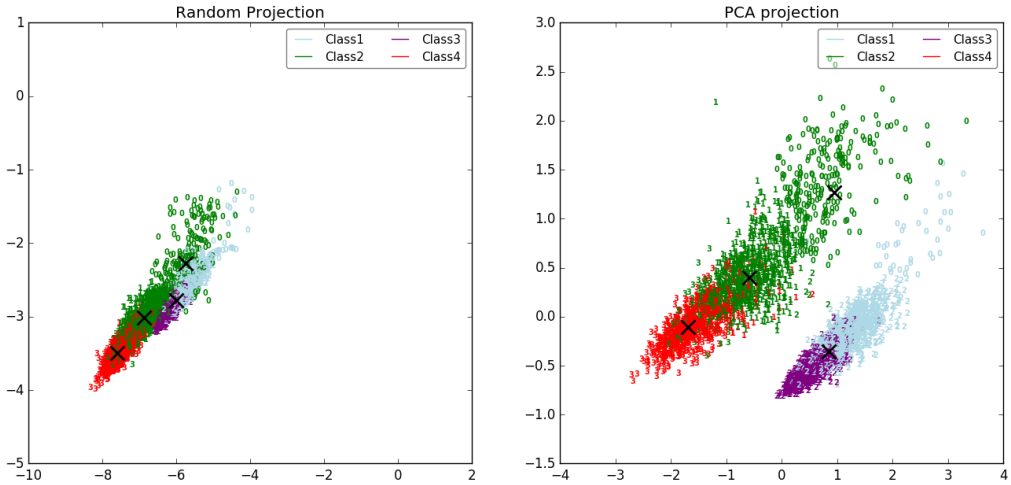


(c) Feature vectors: Serie delle accelerazioni - Lunghezza: 15

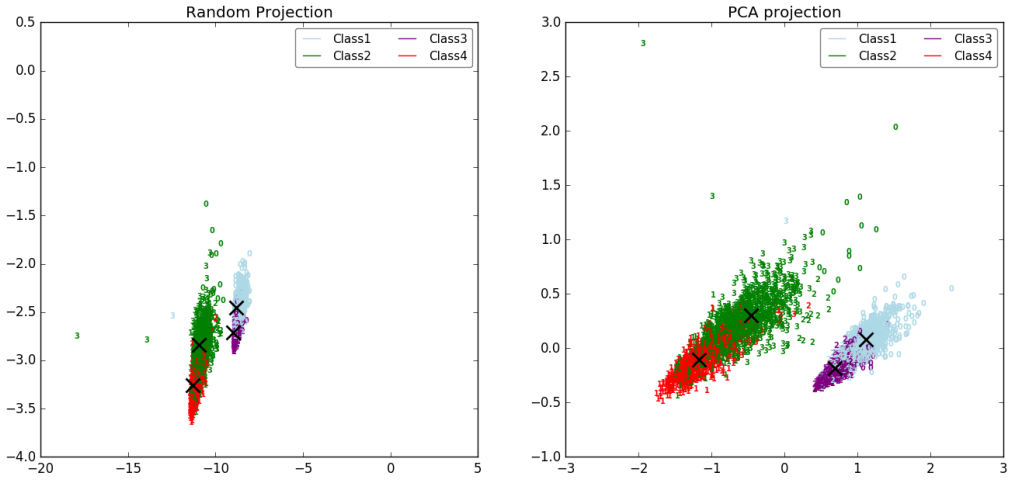
Figura 7.6: Riduzione dimensionale - Confronto fra feature vectors costituiti dalle sole accelerazioni nel caso di ordinamento decrescente



(a) Feature vectors: Serie delle accelerazioni - Lunghezza: 100

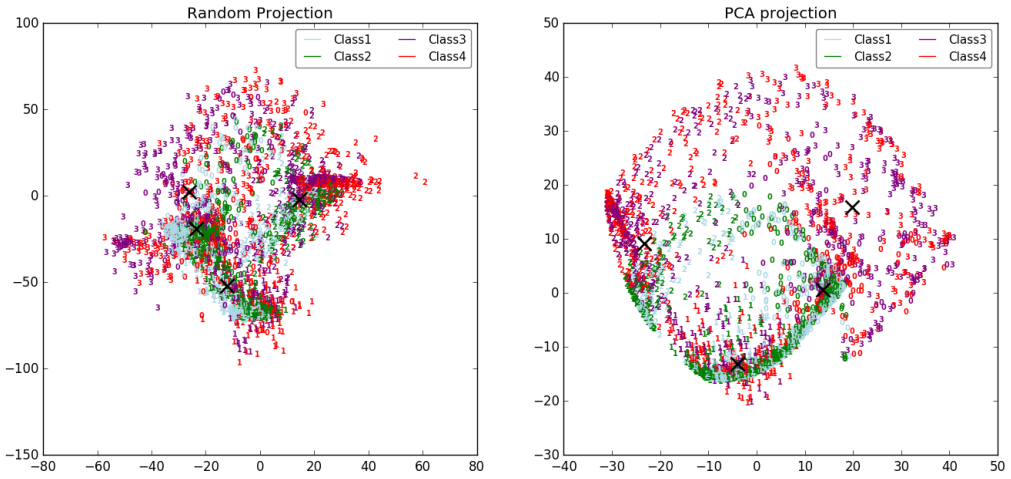


(b) Feature vectors: Serie delle accelerazioni - Lunghezza: 25

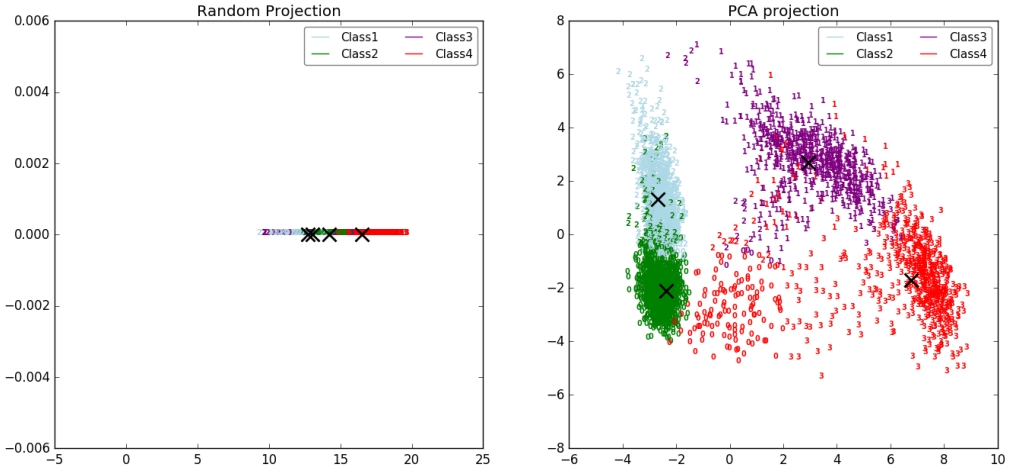


(c) Feature vectors: Serie delle accelerazioni - Lunghezza: 15

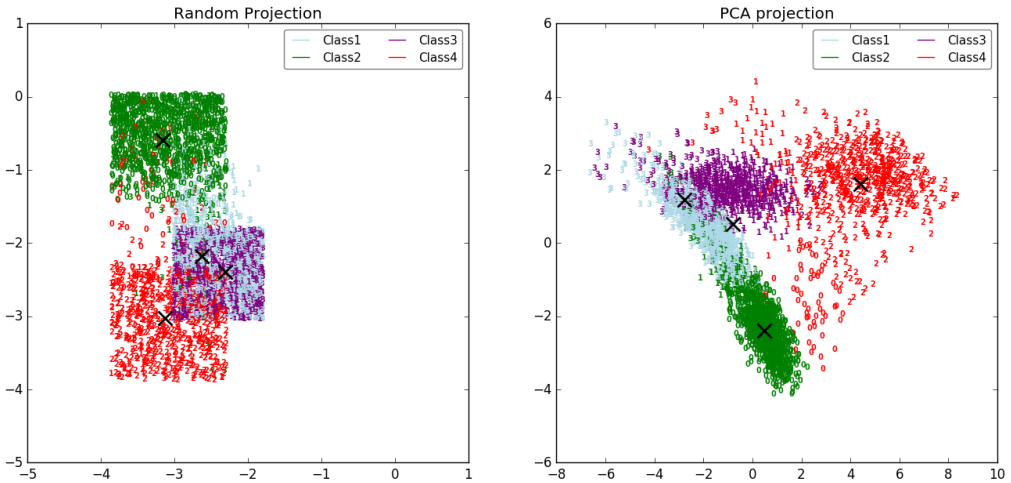
Figura 7.7: Riduzione dimensionale - Confronto fra feature vectors costituiti dalle coppie velocità e accelerazione nel caso senza ordinamento



(a) Feature vectors: Coppie velocità e accelerazione - Lunghezza: 100

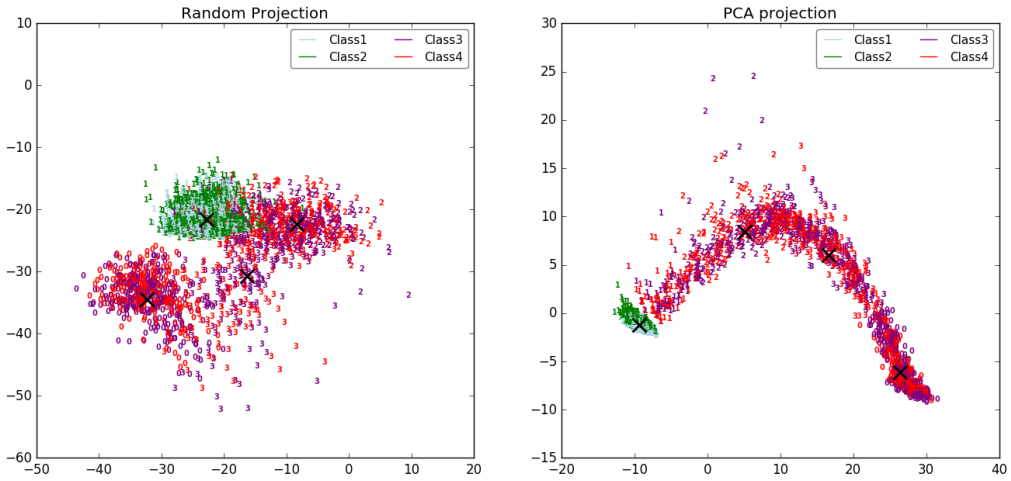


(b) Feature vectors: Coppie velocità e accelerazione - Lunghezza: 20

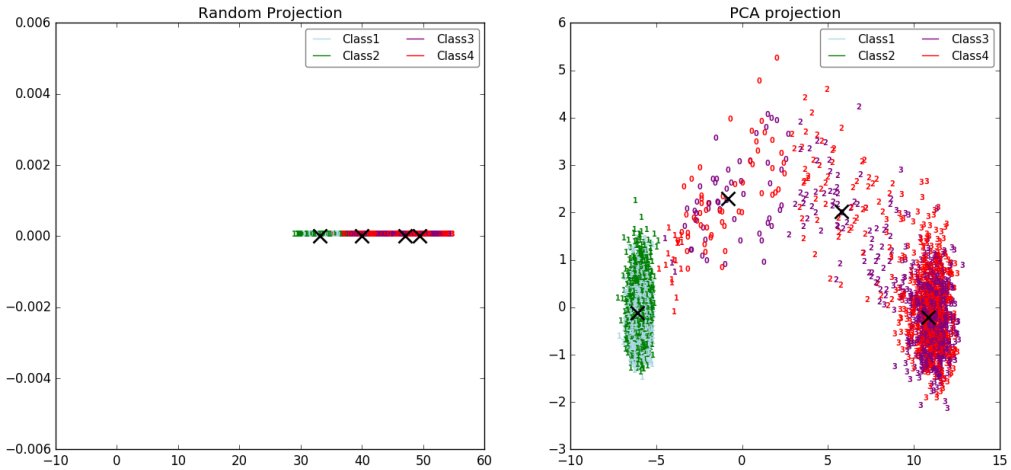


(c) Feature vectors: Coppie velocità e accelerazione - Lunghezza: 15

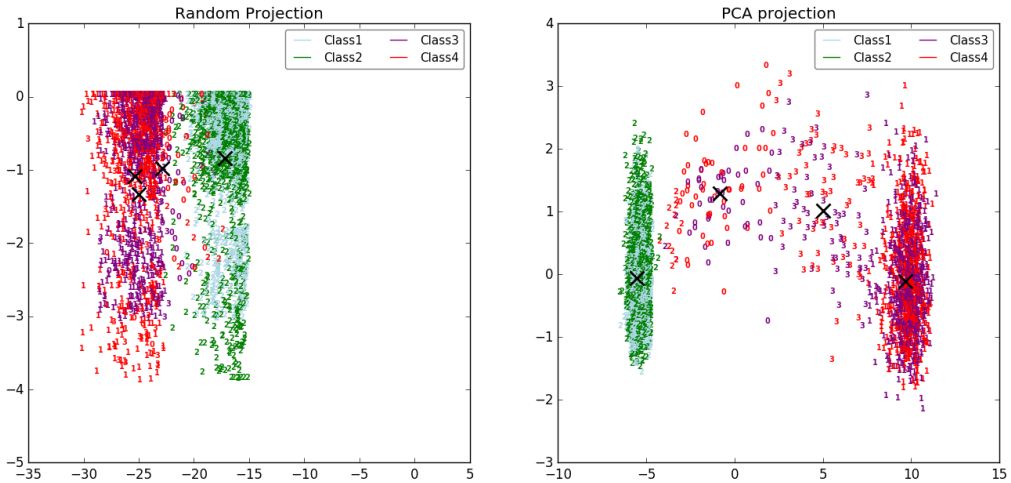
Figura 7.8: Riduzione dimensionale - Confronto fra feature vectors costituiti dalle coppie velocità e accelerazione nel caso di ordinamento decrescente



(a) Feature vectors: Coppie velocità e accelerazione - Lunghezza: 100



(b) Feature vectors: Coppie velocità e accelerazione - Lunghezza: 20



(c) Feature vectors: Coppie velocità e accelerazione - Lunghezza: 15

Capitolo 8

Conclusioni e sviluppi futuri

Dai risultati ottenuti nei tre scenari mostrati possiamo estrarre le seguenti considerazioni:

1. Tipicamente, feature vectors di lunghezza inferiore sono stati in grado di identificare meglio il clustering rispetto a feature vectors di lunghezza maggiore. Da questo possiamo dedurre che uno stile di guida possa essere identificato non dalla totalità delle tracce di un automobilista, ma semplicemente da un loro sottoinsieme piuttosto ridotto

2. L'ordinamento delle tracce può giocare un ruolo importante nella rilevazione di uno stile di guida. In particolare, si è visto di come un ordinamento decrescente abbinato a feature vectors molto corti abbia facilitato l'identificazione del clustering laddove la mancanza di ordinamento otteneva risultati meno efficaci. Da questo possiamo dedurre di come uno stile di guida possa essere definito principalmente dagli eccessi di un automobilista piuttosto che dai suoi comportamenti intermedi e dunque abituali. Si ha infatti che, a livello intuitivo, si potrebbe pensare di come i comportamenti più comuni e dunque meno significativi ai fini del clustering, si andranno a collocare in fondo ad un feature vectors ordinato in senso decrescente. Se poi tale feature vectors venisse anche tagliato ai soli, ad esempio, 5 elementi, allora i comportamenti più comuni verrebbero proprio scartati e non andrebbero ad influenzare negativamente la discriminazione tra le tipologie.

3. Anche l'utilizzo di tracce non ordinate si è rivelato di utilità nella rilevazione di uno stile di guida. In particolare, ciò si è reso evidente nel secondo e nel terzo scenario (rispettivamente Increasing-sf e Merged). Nel caso di Increasing-sf quello che si è

potuto osservare è stato di come, per quel che riguarda la serie delle accelerazioni, la mancanza di ordinamento sia stata più efficace dell'ordinamento in senso decrescente. Nonostante infatti i veicoli avessero tutti la stessa capacità di accelerazione massima e differissero solo per speed factor, l'analisi delle tracce non ordinate si è dimostrata informativa andando a evidenziare una possibile correlazione fra l'aggressività e il modo in cui un veicolo accelera nel tempo. Anche nel caso Merged, dove i veicoli laddove erano simili in caratteristiche fisiche differivano in aggressività e viceversa, si è osservato di come la maggiore efficacia la si abbia avuta sempre nel caso di serie temporali non ordinate. Ciò è in linea con ciò che è stato appena osservato per lo scenario Increasing sf e mostra di come tipologie di veicoli uguali, ma con aggressività diverse tendano a raggiungere le loro capacità massime in modo differente

4. L'ordinamento delle tracce in senso crescente si è dimostrato sempre fallimentare sia con feature vectors lunghi che corti. A livello intuitivo, questo può essere a prova del fatto che comportamenti che non rappresentino eccessi sono comuni a tutti gli automobilisti

5. I dati aggregati, come da aspettative, hanno sempre ottenuto successo ponendosi come limite superiore per i risultati dell'analisi sequenziale. Come anticipato, infatti, nel capitolo 4, velocità massima, accelerazione massima e decelerazione massima sono proprio i parametri direttamente configurabili in SUMO per definire le classi di veicoli e dunque un clustering ottimo su questi parametri può servire come riprova del corretto funzionamento del simulatore.

Un possibile sviluppo futuro è quello di effettuare il clustering su veicoli che abbiano un percorso comune al fine di verificare il loro stile di guida non in modo generico, ma relativamente ad uno stesso percorso specifico. In particolare, a questo scopo è stata già realizzata una classe EdgeVehicle che tenga traccia dei percorsi in edge dei veicoli. Inoltre, il file di configurazione clustering.json permette di ricevere come parametro un path-filter in modo da considerare solo i veicoli che attraversano quel path. In secondo luogo, si potrebbe pensare di avvalersi di Denoising Autoencoder (Vedi Appendice A) al fine di estrarre in modo automatico i vettori di feature a partire da una data serie temporale. In particolare, è stata predisposta una classe Encoder.py, ma non è stata usata in quanto computazionalmente

troppo costosa. Inoltre si potrebbero si potrebbero testare algoritmi di clustering alternativi a k-means Per concludere, sarebbe sicuramente interessante provare ad eseguire il clustering su tracce reali, in modo da verificare l'efficacia degli algoritmi realizzati.

Appendice A

Auto Encoder

A.1 Architettura e funzionamento

Un Auto-Encoder [25] è un particolare tipo di rete neurale. In particolare, a livello di architettura, la forma più semplice di un Auto-Encoder è molto simile a quella di un multilayer perceptron, con un livello di input, un livello di output ed uno o più strati nascosti a connetterli. Si ha però che un Auto-Encoder a differenza di un multilayer perceptron:

1. Ha lo stesso numero di nodi nello strato di Input e nello strato di Output
2. Invece di predire un valore di uscita target Y dati degli input X, è addestrato per ricostruire il suo stesso input X.

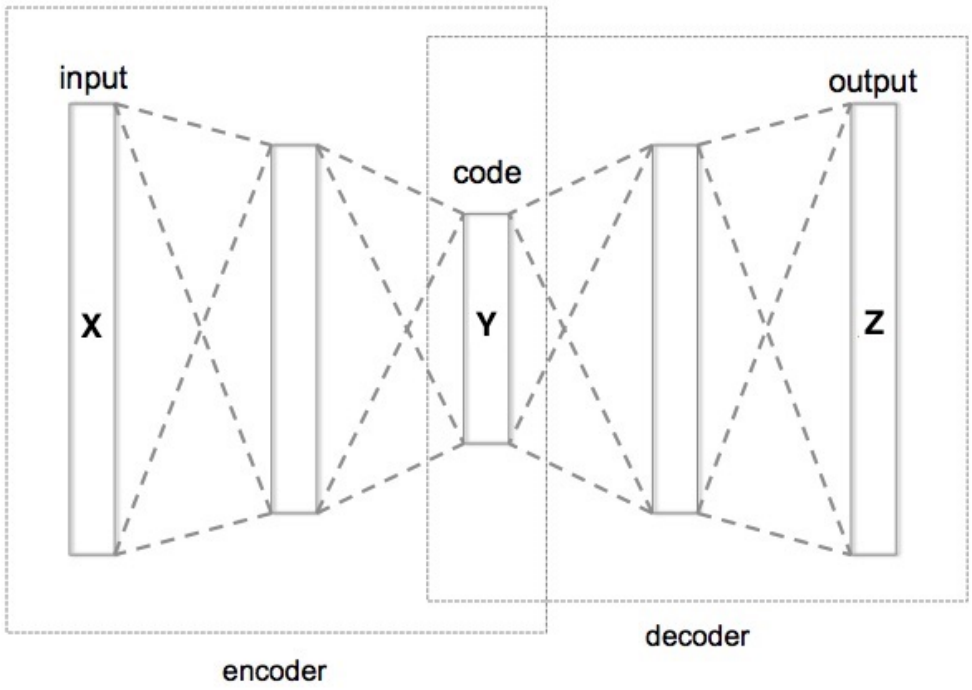
L'idea di base è quella che un Auto-Encoder prenda un input $x \in [o, 1]^d$ e lo codifichi con il processo di encoding nella sua equivalente rappresentazione "nascosta" $y \in [0, 1]^{d'}$ attraverso una codifica deterministica del tipo:

$$y = s(Wx + b)$$

Dove s è una funzione non lineare come ad esempio la sigmoide. La rappresentazione y è anche chiamata "codice" ed è poi mappata all'indietro attraverso il processo di decoding in una ricostruzione z che abbia la stessa forma di x utilizzando una trasformazione simile a quella usata nella fase di encoding:

$$z = s(W'y + b')$$

Figura A.1: Struttura generica Auto-Encoder



Osserviamo dunque di come un Auto-Encoder consista essenzialmente di due parti: l'Encoder e il Decoder (Figura A.1).

In modo opzionale, si può definire che la matrice dei pesi W' sia la trasposta delle matrice W usata nella fase di encoding. In tale caso si avrebbe: $W' = W^T$ e si parla di "tied weights".

A.2 Undercomplete vs Overcomplete

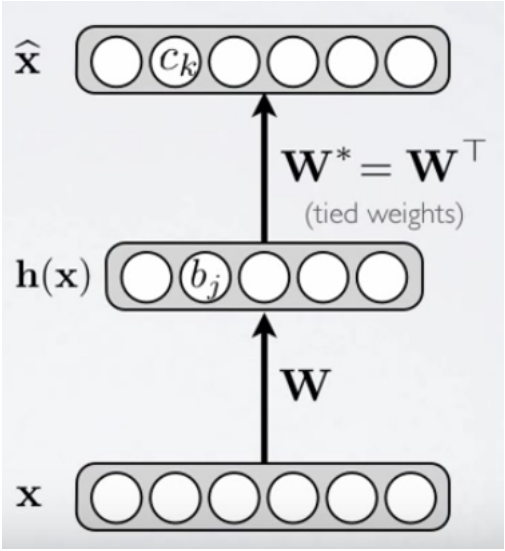
Per concludere, si ha che un Auto-Encoder può avere lo strato nascosto:

1. Undercomplete, ovvero con un numero di nodi inferiore al numero di nodi dello strato di input. In tal caso si ha che tale livello deve eseguire una sorta di "compressione" al fine di poter rappresentare le informazioni dell'input con però un numero inferiore di nodi.

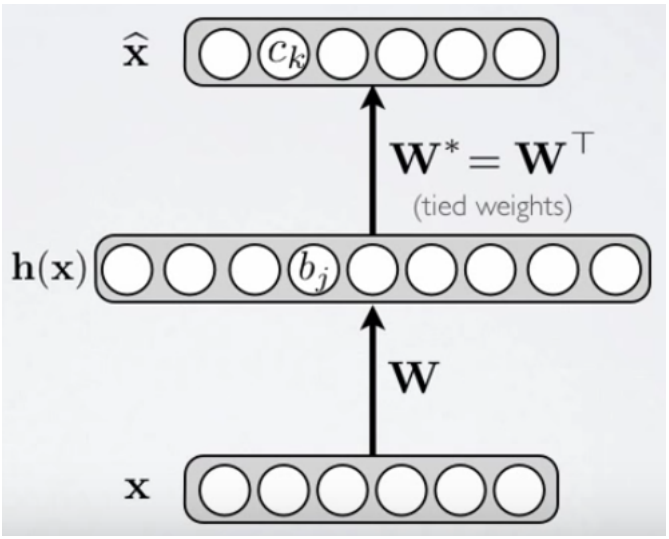
2. Overcomplete, ovvero con un numero di nodi superiore al numero di nodi dello strato di input. In tal caso, non è necessaria alcuna compressione dell'input, ma non vi è alcuna

garanzia che le unità nascoste estraggano informazioni utili dai dati. In particolare, potremmo anche avere che l'Auto-Encoder impari la funzione identità semplicemente copiando l'input.

Figura A.2: Confronto fra Overcomplete e Undercomplete Auto-Encoder



(a) Undercomplete Auto-Encoder



(b) Overcomplete Auto-Encoder

In ogni caso, la speranza è che il codice $Y(X)$ catturi le caratteristiche principali dei dati di input.

A.3 Verso i Denosing Auto-Encoders

Uno dei problemi principali con gli Auto-Encoder è quello citato sopra della possibilità che un Auto-Encoder con un numero n di nodi nel suo strato nascosto pari o superiore al numero di nodi dello strato di input possa semplicemente apprendere la funzione identità.

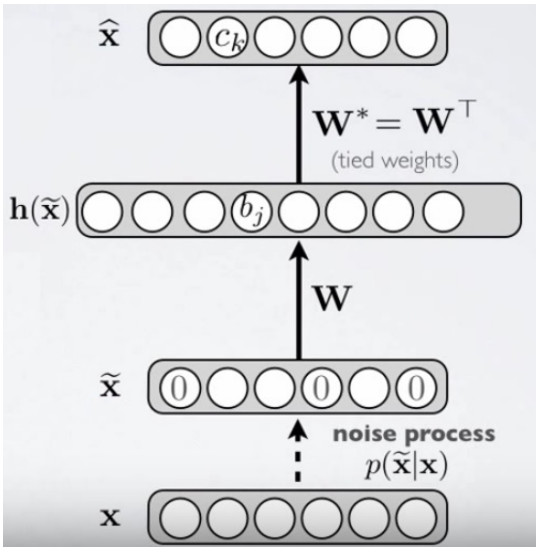
Sorprendentemente, alcuni esperimenti fatti [26] suggeriscono che, nella pratica, Over-complete Auto-Encoders non lineari addestrati con discesa stocastica del gradiente, riescono comunque a produrre rappresentazioni utili attraverso il settaggio opportuno dei pesi nei diversi strati. In alternativa, o in aggiunta al tuning dei pesi, per fare in modo che un Auto-Encoder non apprenda la funzione identità, una strategia usata è quella di aggiungere del rumore ai dati di input prima di farne il coding ottenendo quello che è noto come Denoising Auto-Encoder.

A.4 Denoising Auto-Encoders

A.4.1 Architettura e funzionamento

Come anticipato, un Denoising Auto-Encoder è una versione stocastica di un Auto-Encoder dove l'input utilizzato nella fase di encoding viene prima stocasticamente corrotto attraverso una fase nota come "noise process".

Figura A.3: Sintesi funzionamento Denoising Auto-Encoder



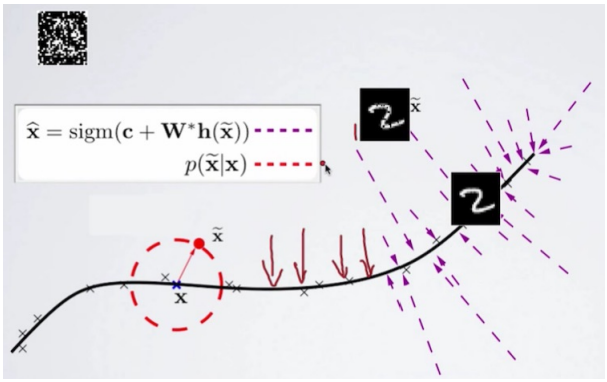
In particolare, tale "noise process" può essere fatto in diversi modi, ad esempio settando alcuni valori dell'input a zero in modo probabibilistico oppure aggiungendo del rumore attraverso una Gaussiana.

Come si può osservare in figura A.3, il funzionamento di un Denoising Auto-Encoder è del tutto uguale a quello di un normale Auto-Encoder con l'unica differenza che prima della fase di encoding viene eseguito il noise process. Inoltre, il Denoising Auto-Encoder, all'atto della ricostruzione, non utilizza per il confronto l'input corrotto, ma bensì quello originale, ovvero, nel caso mostrato in figura A.3 x e $\hat{x}$

A.4.2 Catturare la relazione fra gli input

Come abbiamo visto nel paragrafo precedente, un Denoising Auto-Encoder deve cercare in qualche modo di ricostruire l'input nonostante il fatto che esso possa essere stato corrotto durante il noise process. È importante però osservare però di come ciò possa essere fatto solo a patto che il Denoising Auto-Encoder riesca in un qualche modo a catturare la dipendenza statistica tra gli input stessi.

Figura A.4: Denoising Auto-Encoder - Ricostruzione



In particolare, data la figura A.4 possiamo pensare alla linea mostrata come alla curva che seguono i dati e ai punti fuori da essa come ai dati che sono stati sottoposti al noise-process. Il Denoising Auto-Encoder dovrà dunque ricostruire l'input nel modo corretto riportandoli sulla curva (frecce viola) e per farlo dovrà dunque imparare qual'è la relazione fra i dati stessi.

Appendice B

Dettagli implementativi

B.1 Divisione delle responsabilità

Alla base di questo progetto vi è un principio di ingegneria del software che è quello della divisione delle responsabilità. In particolare, sono stati realizzati moduli distinti che potessero essere il più possibile indipendenti l'uno dall'altro.

Il progetto si compone di due moduli principali:

1. VehiclesGenerator.py –> Tale modulo, a partire da un file di configurazione in formato json, si occupa di generare una serie di percorsi e di eseguire una serie di simulazioni

2. DataAnalyzer.py –> Tale modulo, a partire da un file di configurazione sempre in formato json, si occupa dell'analisi dei dati prodotti dalla simulazione e del loro clustering.

Alla base di tutto vi è poi uno script di utilità setup.py che si occupa di configurare l'ambiente in modo che tutto funzioni correttamente. Nel dettaglio, tale script controlla dapprima che sia stata definita la variabile di ambiente SUMO_HOME senza la quale gli strumenti SUMO non potrebbero funzionare. In seguito, esegue una pulizia dell'ambiente eliminando file relativi a simulazioni precedenti. Per concludere, definisce un file nascosto .definition contenente nella prima riga semplicemente il path della root directory. Tale stratagemma si è rivelato utile sia per verificare se il setup fosse stato eseguito almeno una volta sia per dare un riferimento alla root del progetto ai diversi script che ne avevano necessità.

B.2 Le generazione dei percorsi

Per la generazione dei percorsi inizialmente ci si è avvalsi del modulo randomTrips.py offerto da SUMO. Tale script, infatti, genera dei percorsi in modo casuale ed è configurabile con diverse opzioni. In seguito, per velocizzare le operazioni, si è scelto di definirne un wrapper che ne potesse anche estendere le funzionalità: TripsGenerator.py.

B.2.1 Una generazione casuale

TripsGenerator.py è uno script che a partire da una serie di parametri di configurazione, crea dei processi che si occupino di lanciare in un primo momento randomTrips.py e in seguito un modulo SUMO noto come DUAROUTER che si occupa di assegnare i percorsi generati ai veicoli.

Figura B.1: Dettaglio dello script TripsGenerator.py

```
random_trips_cmd = random_trips + \
        " -n " + self._net_file_path + \
        " -e " + str(n_trips) + \
        " --intermediate " + str(demand["intermediate"]) + \
        " -o " + trips_xml + \
        " --trip-attributes=\"type=\\\"private\\\"\"" + \
        " --additional-file " + self._add_files_path

if demand["use_longer_edges"]:
    random_trips_cmd += " -l"

print "\nCalling random trips: " + random_trips_cmd + "\n"
os.system(random_trips_cmd)

duarouter_cmd = duarouter + \
    " -n " + self._net_file_path + \
    " -t " + trips_xml + \
    " -o " + rou_xml + \
    " --additional-files " + self._add_files_path + \
    " --vtype-output " + vtypes_out + \
    " --end " + str(n_trips) + \
    " --ignore-errors " + \
    " --no-step-log " + \
    " --no-warnings " + \
    " --repair "

print "\nCalling duarouter:", duarouter_cmd, "\n"
os.system(duarouter_cmd)
self.__uniform_routes_time(n_trips, demand["depart_sametime"], demand["period"])
```

Come si può osservare dal codice mostrato in figura B.1, lo script elabora prima il coman-

do randomTrips.py passandogli come parametri la rete, il numero di percorsi da generare (opzione -e), il numero di nodi intermedi minimo che un veicolo deve attraversare prima di uscire dalla simulazione (--intermediate) e altre informazioni di rilievo come ad esempio il path di un file addizionale da usare per la generazione dei tipi. In seguito invoca il modulo DUAROUTER.

Al fine di superare il problema dell'eterogeneità in termini di tempo passato all'interno della rete da parte dei veicoli menzionato nel capitolo 2, inizialmente si è cercato di fare in modo che i veicoli entrassero nel più breve tempo possibile all'interno della simulazione, ma che vi restassero poi a lungo. In particolare, per aumentare il tempo dei veicoli all'interno della simulazione si è incrementato il numero di --intermediate nella chiamata a randomTrips.py. Per fare entrare il maggior numero di veicoli nel più breve tempo possibile, invece, si è dovuti ricorrere ad un piccolo stratagemma. Come si può osservare dal dettaglio dello script TripsGenerator.py in figura B.1, dopo aver chiamato randomTrips.py e duarouter viene invocato il metodo uniform_routes_time. Tale metodo si occupa di uniformare il file *.rou.xml andando a definire ogni quanto debbano partire nuovi veicoli (demand["period"]) e quanti ne debbano partire contemporaneamente (demand["depart_same_time"]). Così facendo è stato possibile incrementare il numero di veicoli per istante di tempo e diminuire i tempi di ingresso dei veicoli nella simulazione.

B.2.2 Il problema della congestione

La strategia adottata di inserire nella rete tutti i veicoli nel più breve tempo possibile in modo da uniformare la dimensione delle serie temporali estratte ha avuto però la controindicazione involontaria di non permettere di simulare un grande numero di veicoli senza andare a congestionare la rete. È risultato subito evidente e visibile tramite la sumo-gui, di come l'inserimento di oltre 1000 veicoli in meno di 200 timestep comportasse una immediata congestione della rete rendendo lo scenario inadatto al clustering.

In un primo momento la soluzione adottata è stata quella di inserire un numero relativamente basso di veicoli (dalle 300 alle 800 unità), ma con un numero di --intermediate alto (maggiore di 5) al fine da simulare questi veicoli per un periodo di tempo più lungo come ad esempio 7200 secondi (2 ore). Osserviamo di come questa sia una pseudo-soluzione, in quanto il tempo di simulazione non è realmente scalabile. Il numero di percorsi validi

Figura B.2: Dettaglio metodo uniform_routes_time

```
counter = 0
vid = 0
depart_time = 0.00
for event, vehicle in rou_file_tree:
    vehicle.attrib['depart'] = str(depart_time) + "0"
    vehicle.attrib['id'] = str(vid)

    vid += 1
    counter += 1
    if counter % depart_sametime == 0:
        depart_time += period

rou_file = open(rou_file_to_parse, "r")
header = ""
line = rou_file.readline().rstrip()
while line != "-->":
    header += line + "\n"
    line = rou_file.readline().rstrip()
header += "-->\n\n"
rou_file.close()

output_xml = open(rou_file_to_parse.replace("rou.xml", "uniformed.rou.xml"), "w")
output_xml.write(header + et.tostring(rou_file_tree.root))
output_xml.close()
```

generati da randomTrips.py, infatti, cala con l'aumentare del numero dei nodi intermedi per il quale si costringe i veicoli a passare. Inoltre, i dati raccolti saranno sempre relativi a quel set di veicoli che semplicemente avranno delle serie più lunghe. La soluzione che è stata adottata in seguito al fine di simulare un numero maggiore di veicoli, è stata quella di realizzare un altro script che permettesse di avere una generazione più bilanciata dei percorsi: TripsGeneratorVPH.py.

B.2.3 Una generazione bilanciata

TripsGeneratorVPH.py è uno script mutuato da uno script esistente realizzato da Federico Caselli per un progetto dell'università di Bologna e che, a differenza di TripsGenerator.py permette di generare dei percorsi a partire da un target di veicoli orario. In altre parole, tale script oltre a permettere di definire per quanto tempo simulare, permette anche di definire approssimativamente quanti veicoli dovranno entrare all'interno della rete per ogni ora di simulazione. Un altro dei vantaggi offerti da TripsGeneratorVPH.py è che la generazione dei percorsi non è più fatta in modo casuale, ma sfrutta delle "routes" estratte da percorsi reali nel reticolo di Andrea Costa rendendo la simulazione maggiormente realistica.

B.2.4 I file di configurazione

A seconda della modalità prescelta, sono stati dunque definiti due file di configurazioni differenti: il file configuration.json per la modalità a numero di veicoli crescenti e il file configurationVPH.json per la modalità che prevede l'inserimento di un numero costante di veicoli per ogni ora di tempo simulato. Andiamo ora a dettagliare la struttura dei file di questi file.

configuration.json

Il file di configurazione configuration.json permette di definire una serie di simulazioni da eseguire e per ognuna di esse definisce dei parametri generali e una o più demand che vanno a definire delle caratteristiche specifiche. Nel dettaglio, come parametri generali che caratterizzano una simulazione si sono scelti i valori mostrati in tabella B.1.

Tabella B.1: Descrizione degli attributi di simulazione per il file configuration.json

Parametro	**Descrizione**
net_name	Il nome della rete da utilizzare.
net_file	Il path dove è situato il file della rete da utilizzare (*.net.xml)
cfg_file	Il path dove è situato il file di configurazione *.sumo.cfg
add_files	Array di file addizionali che ne definisce i path. Il primo deve essere quello relativo alle tipologie di veicoli.
vehicle_storage	Definisce come i veicoli dovranno essere salvati alla fine della simulazione. Le possibilità sono EasyVehicle e EdgeVehicle.
simtime	Il tempo di simulazione espresso in secondi.
lazy	Definisce attraverso un attributo booleano se la simulazione dovrà salvare i veicoli in modo lazy o meno.
initial_n_trips	Definisce il valore iniziale per i percorsi da generare
end_n_trips	Definisce il valore finale per i percorsi da generare
step	Definisce lo step di incremento per giungere a end_n_trips a partire da begin_n_trips.

Per quel che riguarda invece i parametri per ogni singola demand, si è scelto di considerare i valori mostrati in tabella B.2.

Tabella B.2: Descrizione degli attributi per singola demand per il file configuration.json

Parametro	**Descrizione**
intermediate	Numero minimo di nodi intermedi che un veicolo deve attraversare prima di uscire dalla simulazione.
period	Numero di veicoli da generare per secondo
depart_sametime	Numero di veicoli con lo stesso istante di partenza
use_longer_edges	Definisce attraverso un booleano se per i percorsi assegnati ai veicoli, randomTrips.py dovrà o meno dare priorità alla scelta di edge più lunghi

configurationVPH.json

Il file configurationVPH.json è stato mutuato da un file di configurazione creato da Federico Caselli nell'ambito di un progetto universitario. In particolare, tale file definisce dei parametri generali per la simulazione analoghi a quelli definiti da configuration.json, ma a differenza di quest'ultimo definisce anche una serie di informazioni relative alle routes da usare nella generazione dei percorsi e quali di esse usare come inizio e terminazione. Per concludere, tale file permette di specificare delle demand nelle quali è possibile indicare quanti veicoli inserire per ogni ora di tempo simulato e per quante ore mantenere questo regime. Tali demand vengono poi aggregate in un unico file *.rou.xml. In altre parole, definendo due demand rispettivamente da 3600 vph e 1800 vph per un ora, si otterrà un file *.rou.xml dove nella prima ora partiranno 3600 veicoli, mentre nella seconda ne partiranno 1800.

B.3 L'esecuzione della simulazione

Per quel che riguarda l'esecuzione della simulazione, essa è svolta dal modulo TraciParser.py che sfrutta le API di traci al fine di avviare una simulazione e di estrarne in tempo reale le informazioni considerate rilevanti. In particolare, come si può vedere dal codice in

figura B.3, traci offre un metodo start che prende come parametro un comando sumo che specifichi tramite opzione -c il file di configurazione *.sumo.cfg da usare.

Figura B.3: Dettaglio metodo simulate

```
def __simulate(self, lazy):
    sumo_cmd = [sumo, "-c", self._copy_config_file_path]
    traci.start(sumo_cmd)

    if lazy:
        self.__lazy_simulate()
    else:
        self.__eager_simulate()
```

In seguito, la simulazione può essere poi eseguita nelle due modalità lazy o eager menzionate nella sezione 2.5.
TraciParser.py è stato realizzato fin da principio per essere facilmente personalizzabile in quanto inizialmente non si era certi di quali sarebbero potuto essere le informazioni rilevanti da estrarre. In particolare, è stato definito un attributo vehicle_storage che definisce quali informazioni dovrà salvare il parser. Nel corso del progetto, infatti, sono state definite alcune classi di supporto al fine di poter modellare al meglio l'ambiente di simulazione.

1. *Vehicle* –> È una classe astratta e dunque non istanziabile che si pone alla base della gerarchia dei veicoli. Un Vehicle ha semplicemente un id e un tipo

2. EasyVehicle –> È la classe di supporto più semplice e quella sulla quale sono stati condotti la maggioranza degli esperimenti. Un EasyVehicle estende Vehicle aggiungendo una serie di velocità e di accelerazioni istantanee e una serie di speed factor. Inoltre tale classe definisce una serie di modalità per ottenere, a partire dai suoi attributi, dei features vectors utili per il clustering

3. EdgeVehicle –> Classe di supporto che aggiunge alle caratteristiche dell'EasyVehicle la possibilità di tenere conto del percorso seguito dal veicoli in termini di Edge attraversati

4. Edge –> Classe di supporto che modella un Edge

Il TraciParser.py in modalità EasyVehicle, dunque, si occupa di estrarre per ogni veicolo la serie delle sue velocità e accelerazioni istantanee e la serie dei suoi speed factor.

B.4 L'analisi della simulazione

Al fine di valutare la coerenza dei risultati delle simulazioni eseguite con le tipologie dei veicoli definiti si è deciso di realizzare uno script SimDataAnalyzer.py che fosse in grado, a partire proprio da quei risultati, di mostrare se esistesse e in caso affermativo di quale fosse la correlazione fra simulazione eseguita e file addizionale dei tipi di veicolo usato tramite istogrammi. Nel dettaglio, SimDataAnalyzer.py si occupa di definire dei grafici per le caratteristiche principali definite nel file addizionale vtypes.add.xml e per la distribuzione dei veicoli simulati.

B.4.1 La simulazione

Una volta caricato il file di configurazione, eseguito il setup ed eseguita la generazione dei percorsi (rispettivamente scelte 0 e 1 in figura B.4), VehiclesGenerator.py si occupa di chiedere all'utente se desidera o meno lanciare la simulazione.

Figura B.4: Esempio di interazione con lo script VehiclesGenerator.py

```
MacBook-Pro-di-Alessandro-3:src Alessandro$ python VehiclesGenerator.py
0 - Load TripsGenerator configuration
1 - Load TripsGenerator VPH configuration
exit - To quit

Insert the number of your choice: 0

CONFIGURATION configuration.json
0 - Setup and reload configuration
1 - Generate random trips
2 - Simulate
3 - Change configuration
exit - To quit

Insert the number of your choice:
```

In caso di risposta affermativa, provvede ad usare lo script TraciParserMP.py (versione multi processo dello script TraciParser.py) per eseguire quante più simulazioni possibili in parallelo. I risultati di tali simulazioni sono poi opportunamente spostati in una cartella SIMULATIONS in modo da poter essere poi usati opportunamente dal modulo di clustering.

B.5 L'analisi dei dati

L'analisi dei dati, come anticipato in precedenza, è svolta dal modulo DataAnalyzer.py che si occupa dell'analisi dei dati prodotti dalla simulazione e del loro clustering. Nel dettaglio, quando viene lanciato chiede all'utente quale tipo di analisi sui dati prodotti dalla simulazione vuole eseguire.

Figura B.5: Esempio di interazione con lo script DataAnalizer.py

```
MacBook-Pro-di-Alessandro-3:src Alessandro$ python DataAnalyzer.py
0 - Sim Data analysis
1 - Cluster Data Analysis
2 - Execute all
exit - To quit

Insert the number of your choice:
```

A seconda della scelta userà lo script SimDataAnalyzer.py oppure lo script ClusterDataAnalyzer.py (rispettivamente scelta 0 e scelta 1 in figura B.5). I risultati dell'elaborazione sono poi trasferiti in una cartella relativa alla simulazione analizzata.

Bibliografia

[1] https://en.wikipedia.org/wiki/Motor_vehicle

[2] http://www.acea.be/statistics/tag/category/passenger-car-fleet-per-capita

[3] http://www.who.int/violence_injury_prevention/road_safety_status/2015/en/

[4] http://www.istat.it/it/archivio/194394

[5] https://paginas.fe.up.pt/ prodei/dsie11/images/pdfs/s2-3.pdf

[6] http://vision-traffic.ptvgroup.com/en-us/products/ptv-vissim/

[7] http://www.paramics-online.com/

[8] https://www.aimsun.com/aimsun/

[9] https://its.mit.edu/software/mitsimlab/technical-information

[10] https://its.mit.edu/software/mitsimlab

[11] http://www.dlr.de/ts/en/desktopdefault.aspx/tabid-9883/16931_read-41000/

[12] http://sumo.dlr.de/wiki/MESO

[13] http://www.ijcsi.org/papers/IJCSI-9-1-3-115-119.pdf

[14] https://pdfs.semanticscholar.org/b865/ff075f224e5eed9b548f4f98c21410539075.pdf

[15] http://sumo.dlr.de/wiki/Definition_of_Vehicles,_Vehicle_Types,_and_Routes

[16] http://sumo.dlr.de/wiki/Definition_of_Vehicles,_Vehicle_Types,_and_Routes #Speed_Distributions

[17] https://en.wikipedia.org/wiki/Cluster_analysis

[18] http://ilpubs.stanford.edu:8090/778/1/2006-13.pdf

[19] http://scikit-learn.org/stable

[20] http://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html

[21] http://scikit-learn.org/stable/modules/clustering.html#clustering-performance-evaluation

[22] http://scikit-learn.org/stable/modules/generated/sklearn.metrics.homogeneity_score.html#sklearn.metrics.homogeneity_score

[23] http://scikit-learn.org/stable/modules/generated/sklearn.metrics.completeness_score.html#sklearn.metrics.completeness_score

[24] http://scikit-learn.org/stable/modules/manifold.html

[25] http://deeplearning.net/tutorial/dA.html#autoencoders

[26] Bengio, P. Lamblin, D. Popovici and H. Larochelle, Greedy Layer-Wise Training of Deep Networks, in Advances in Neural Information Processing Systems 19 (NIPS'06), pages 153-160, MIT Press 2007.