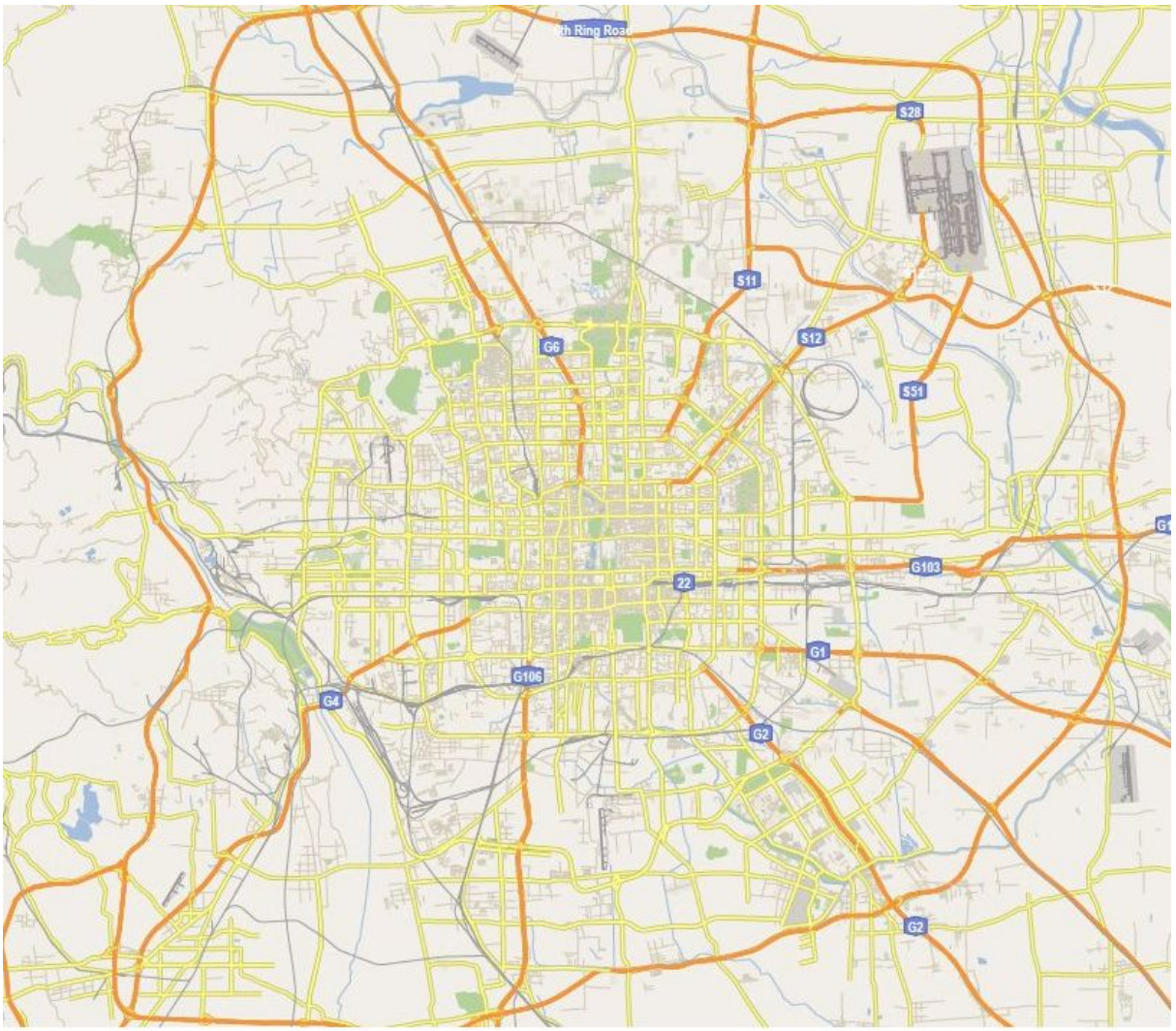


Relazione

Progetto Data Technology e Machine Learning

Pietro Colombo 793679
Costantino Carta 808417
Carlo Radice 807159

Introduzione al progetto

Il progetto sviluppato dal gruppo si pone l'obiettivo di predirre la tipologia del mezzo di trasporto utilizzato per effettuare un determinato percorso.
Il lavoro è stato suddiviso in diverse fasi nelle quali ci si è concentrati ad affrontare una specifica richiesta progettuale.

Nella fase iniziale è stata effettuata la ricerca del dataset che meglio si adeguava al nostro caso di studio. Dopo un'attenta valutazione di diversi dataset, si è quindi scelto di utilizzare il dataset Geolife Trajectories 1.3 che meglio rappresenta i dati utili ad effettuare la nostra predizione.
Inizialmente come secondo dataset si è pensato di usare il dataset OpenStreetMap, disponibile gratuitamente online. Esso contiene dati sulla maggior parte di "vie" (terra, acqua, aria) percorribili del mondo ed è stato utilizzato per recuperare il tipo di "via" data dalle coordinate GPS, tuttavia per conoscere il nome delle città, degli stati e delle regioni è stato necessario utilizzare bing in quanto open street map da questo punto di vista è risultato carente.

Successivamente ci siamo concentrati sulla parte di machine learning.
Sono stati sviluppati due modelli di classificazione, svm e naive bayes, per effettuare la predizione della tipologia del mezzo utilizzata in un determinato percorso. Si sono quindi calcolate le stime delle misure di performance.

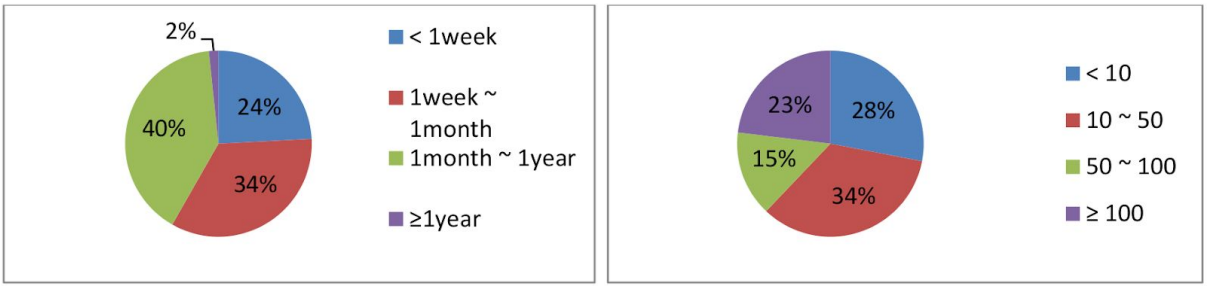
Nella fase finale sono state elaborate le conclusioni del progetto.

Descrizione e analisi iniziale del dataset

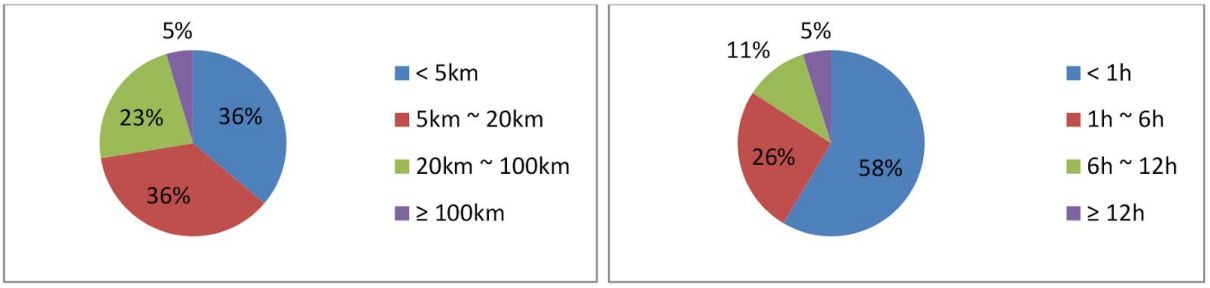
Il primo dataset utilizzato prende il nome di **Geolife Trajectories 1.3**.
Il dataset è stato sviluppato da Microsoft Research Asia collezionando i dati delle traiettorie (o percorsi) GPS rappresentate da 24.876.978 di punti registrati da 182 utenti nel periodo da aprile 2007 ad agosto 2012. Ogni cartella del dataset contiene i file delle traiettorie GPS, in formato PLT, di un singolo utente. Ogni file PLT contiene una singola traiettoria. Il suo identificativo è dato dal tempo di inizio (ore:minuti:secondi giorno:mese:anno) della registrazione. Il tempo viene preso considerando lo standard GMT.
Nel file, per ogni posizione sono definiti i seguenti campi:

1. latitudine in gradi decimali
2. longitudine in gradi decimali
3. questo campo è posto a 0 per tutti i punti (non viene usato per questo dataset)
4. altitudine in piedi (-777 indica che non è presente un valore)
5. data espressa come numero di giorni che sono passati dal 30/12/1899
6. data del giorno nel formato ‘anno-mese-giorno’ come stringa
7. orario del giorno nel formato ‘ora-minuto-secondo’ come stringa

La traiettoria GPS viene rappresentata da una sequenza di punti, i quali contengono le informazioni di latitudine, longitudine e altitudine. Ad ogni punto viene associato il tempo di acquisizione. Il dataset contiene 17.621 traiettorie, le quali sommate arrivano ad ottenere una distanza di 1.292.951 chilometri e una durata di 50.176 ore. Il 91,5% delle traiettorie sono state registrate con una rappresentazione densa dei punti, ovvero è presente un punto ogni 1~5 secondi o 5~10 metri. Per le restanti traiettorie i punti vengono presi ad intervalli di decine di secondi. Il dataset è distribuito in diverse aree, ma maggior parte delle traiettorie è stata registrata in Cina e precisamente a Pechino. La seguente figura rappresenta le zone nel mondo rappresentate nel dataset.



Il grafico a sinistra rappresenta la partizione degli utenti in base alla durata del periodo in cui sono stati sottoposti a raccolta dati, quello a destra contrariamente riporta la distribuzione degli utenti secondo il numero di traiettorie effettuate e quindi raccolte nel dataset.



Il grafico a sinistra rappresenta la distribuzione delle traiettorie secondo la loro distanza, mentre quello a destra riporta la distribuzione delle traiettorie per effettiva durata.

Inoltre per 73 su 182 utenti sono presenti delle label che rappresentano il mezzo di trasporto utilizzato durante le varie traiettorie. Esse sono identificate dal file label.txt contenuto nella cartella di ciascun utente. I possibili mezzi di trasporto sono: walk, bike, bus, car, subway, train, airplane, boat, run, motorcycle. La presenza di questa indicazione sul mezzo di trasporto utilizzato da parte di un determinato utente in una particolare traiettoria si rivela fondamentale allo scopo del progetto, in particolare per quanto concerne la fase di machine learning.

Analisi iniziale consistenza e completezza

Prima ancora di trasformare i dati in una tabella csv, come analisi iniziale, si è effettuato un parsing del dataset iniziale, nel quale è stata controllata ciascuna cartella, allo scopo di eliminare gli utenti senza la label che identifica il mezzo tramite il quale è stata effettuata una traiettoria. Infatti con l'obiettivo di predirre, tramite apprendimento supervisionato, la tipologia di mezzo utilizzata in una determinata traiettoria, i dati senza questa informazione fondamentale (non recuperabile in altro modo) risultano inutili e inutilizzabili, sia in fase di training sia in fase di test. In quanto sia per istruire un modello sia per testare le performance necessitiamo di avere la realizzazione della variabile oggetto di previsione.

Consistenza

Contemporaneamente viene effettuato un controllo di *consistenza* sui dati, in particolare sulle posizioni geografiche. Infatti per le traiettorie etichettate, vengono controllate le posizioni espresse dalle latitudini e longitudini, se non consistenti, ovvero senza valori accettabili (latitudine maggiore di 90° o minore di -90° oppure longitudine maggiore di 180° o minore di -180°), vengono eliminati, in quanto non integrabili in altro modo. Vengono anche

eliminati il terzo campo, che contiene il valore 0 per ogni punto, e il campo alla posizione 5, che contiene la data come numero di giorni a partire dal 30/12/1889, perchè rappresentano informazioni inutili per effettuare la predizione.

Completezza

In fine se ci sono punti rappresentati erroneamente più volte vengono cancellati, allo scopo di eliminare eventuali ridondanze nel dataset.
L'analisi di *completezza* si pone lo scopo di andare a studiare la presenza di valori nulli nelle tuple, nelle colonne e nelle intere tabelle del dataset. Il risultato di questa analisi ha mostrato come in tutto il dataset non ci siano valori nulli.
Su questo dataset selezionato e analizzato viene creata la prima tabella csv.
La tabella contiene una serie di attributi contenenti quindi:

- **Latitudine**: espressa per ogni punto di ogni percorso
- **Longitudine**: espressa per ogni punto di ogni percorso
- **Altitudine**: espressa per ogni punto di ogni percorso se presente. -777 altrimenti
- **Date_Time:** contiene la data di acquisizione della posizione nel formato 'anno-mese-giorno ora-minuto-secondo'
- **Id_user**: identificativo dell'utente
- **Id_perc**: identificativo del percorso
- **Label**: etichetta che rappresenta il mezzo di trasporto utilizzato

Ogni tupla del dataset esprime un punto di un percorso di un utente. Per ogni punto di un determinato percorso viene associato il valore della label che si riferisce a quel percorso e l'utente che lo ha effettuato.
Grazie alla presenza dell'attributo Date_Time si possono facilmente confrontare le date.
Questa è considerabile come la seconda versione del dataset.
In seguito è stata creata una tabella da 2.966.060 punti. rappresentante tutti i percorsi di tutti gli utenti

	Latitude	Longitude	Altitude	Date_Time	Id_user	Id_perc	Label
2	41.741415	86.186028	-777	2008-03-31 16:00:08	10	20080331160008.plt	taxi
3	41.737063	86.17947	-777	2008-03-31 16:01:07	10	20080331160008.plt	taxi
4	41.734105	86.172823	-777	2008-03-31 16:02:07	10	20080331160008.plt	taxi
5	41.73911	86.166563	-777	2008-03-31 16:03:06	10	20080331160008.plt	taxi
6	41.744368	86.159987	-777	2008-03-31 16:04:05	10	20080331160008.plt	taxi
7	41.744513	86.159808	-777	2008-03-31 16:05:04	10	20080331160008.plt	taxi
8	41.748142	86.15533	-777	2008-03-31 16:06:03	10	20080331160008.plt	taxi
9	41.74964	86.153458	-777	2008-03-31 16:07:02	10	20080331160008.plt	taxi
10	41.754737	86.148085	-777	2008-03-31 16:08:02	10	20080331160008.plt	taxi

Aggiunta di features al dataset

Non avendo a disposizione abbastanza features per effettuare la predizione abbiamo deciso di aggiungerne di nuove relative ad ogni percorso di ogni utente.
Nello specifico le features da noi calcolate sono:

- **distance**: rappresenta la distanza tra il punto considerato e il suo precedente nella traiettoria (espressa in metri)

- **delta_time**: intervallo di tempo tra il punto considerato e il suo precedente nella traiettoria (espresso in secondi)
- **vel**: velocità calcolata in un punto utilizzando distance e delta_time (espressa in metri al secondo)
- **angle**: angolo tra il nord e le due coordinate di un punto nella traiettoria

Il nuovo dataset ha quindi la seguente struttura.

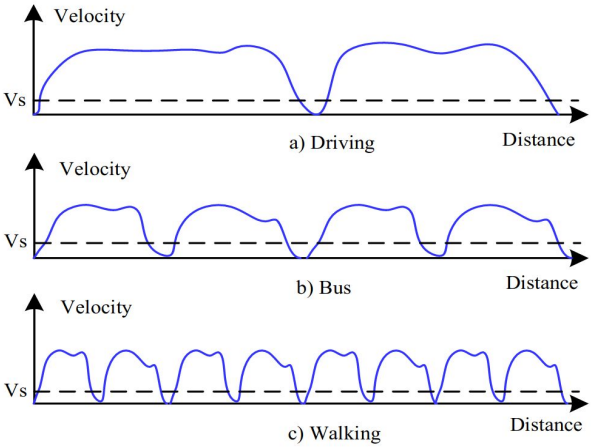
	Latitude	Longitude	Altitude	Date_Time	Id_user	Id_perc	Label	distance	vel	delta_time	angle
2	41.741415	86.186028	-777	2008-03-31 16:00:08	10	20080331160008.plt	taxi	0	0	0	0
3	41.737063	86.17947	-777	2008-03-31 16:01:07	10	20080331160008.plt	taxi	728.881331879734	12.3539208793175	59	3.9855274994827
4	41.734105	86.172823	-777	2008-03-31 16:02:07	10	20080331160008.plt	taxi	643.217261944878	10.7202876990813	60	4.1746946851854
5	41.73911	86.166563	-777	2008-03-31 16:03:06	10	20080331160008.plt	taxi	761.729237892239	12.910665049021	59	5.53229687721404
6	41.744368	86.159987	-777	2008-03-31 16:04:05	10	20080331160008.plt	taxi	800.179842176798	13.5623702063864	59	5.53237191568031
7	41.744513	86.159808	-777	2008-03-31 16:05:04	10	20080331160008.plt	taxi	21.9331839621699	0.371748880714745	59	5.53884900998576
8	41.748142	86.15533	-777	2008-03-31 16:06:03	10	20080331160008.plt	taxi	548.817928919407	9.30199879524419	59	5.53910423790785
9	41.74964	86.153458	-777	2008-03-31 16:07:02	10	20080331160008.plt	taxi	227.873820912181	3.86226815105391	59	5.53279890255922
10	41.754737	86.148085	-777	2008-03-31 16:08:02	10	20080331160008.plt	taxi	721.241419270438	12.020690321174	60	5.6168054193144

Con questa terza versione del dataset diventa possibile a partire dalle n tuple che identificano un singolo percorso di un utente (diviso in n posizioni), ottenere una singola tupla, riducendo notevolmente la dimensione del nuovo dataset fino ad avere 5.514 tuple.
La tupla identificante il percorso avrà una serie di nuove features calcolate a partire dai suoi punti.
La struttura è la seguente:

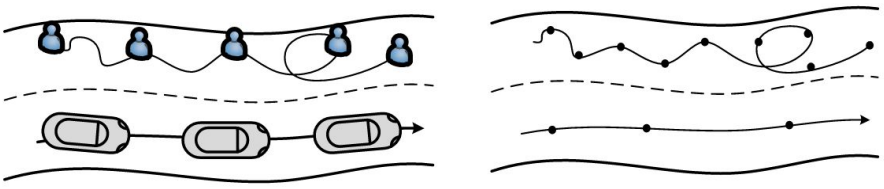
- **Id_user**: identifica l'utente
- **Id_perc**: identifica il percorso dell'utente
- **label**: identifica il mezzo di trasporto utilizzato per il percorso
- **longitudeStart**: identifica la longitudine del primo punto calcolato nel percorso
- **latitudeStart**: identifica la latitudine del primo punto calcolato nel percorso
- **longitudeEnd**: identifica la longitudine dell'ultimo punto calcolato nel percorso
- **latitudeEnd**: identifica la latitudine dell'ultimo punto calcolato nel percorso
- **TimeStart**: identifica l'istante di tempo in cui è stato acquisito il primo punto calcolato nel percorso
- **TimeEnd**: identifica l'istante di tempo in cui è stato acquisito l'ultimo punto calcolato nel percorso
- **distanceTotal**: identifica la distanza totale percorsa (espressa in metri)
- **time_total**: identifica la durata in termini di tempo del percorso (espressa in secondi)
- **n777**: identifica il numero di punti in un percorso che hanno il valore dell'altitudine impostato a -777, e di cui non è quindi possibile conoscere la reale altitudine
- **npoints**: identifica il numero totale di punti calcolati nel percorso
- **vel_avg**: identifica la velocità media del percorso calcolata facendo la divisione tra distance_Total e time_total (espressa in metri al secondo)
- **vel_max**: identifica la velocità massima raggiunta nel percorso (espressa in metri al secondo)
- **altitudeAvg**: identifica l'altitudine media del percorso. Viene calcolata come la somma di tutti i punti con il valore dell'altitudine diverso da -777 diviso il numero di quei punti. Viene assunto il valore NA se non è presente la misura (espressa in metri)
- **altitudeMax**: identifica l'altitudine massima del percorso. Viene assunto il valore -236,5248 se non è presente la misura (viene espressa in metri)
- **vcr**: identifica il velocity change rate, ovvero il rapporto tra numero di punti GPS con variazione di velocità, rispetto al punto precedente, al di sopra di una certa soglia per

unità di distanza, e la distanza totale del percorso. Rapporti diversi indicano differenze tra mezzi di trasporto.

- **sr**: identifica lo stop rate, ovvero il rapporto tra il numero di punti GPS con velocità inferiore ad una certa soglia per unità di distanza, e la distanza totale del percorso. Il grafico mostra un esempio per tre casi diversi, dove Vs è la soglia. Se il mezzo di trasporto è 'walk' è più probabile che l'utente si fermi più spesso rispetto a 'car'.

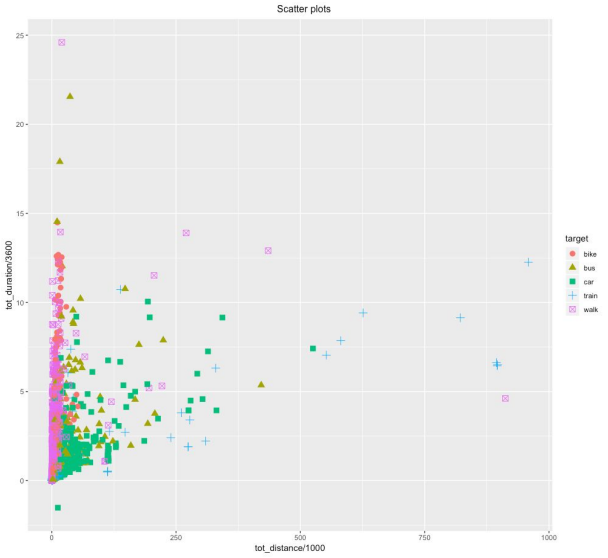


- **hcr**: identifica l'heading change rate, ovvero il rapporto tra il numero di punti GPS in cui un utente cambia direzione con un angolo superiore ad una certa soglia, e la distanza totale del percorso. La figura mostra il caso in cui si è a piedi o in un veicolo. La traiettoria di un utente con mezzo 'walk' è meno lineare e quindi avrà un valore di hcr maggiore rispetto a 'car'.



Analisi di consistenza

Dopo l'aggiunta delle feature abbiamo effettuato dei test sulla veridicità delle velocità relative alle varie classi.
Tramite il seguente grafico si è notato che per la classe 'walk' alcune velocità sono da considerarsi inconsistenti poiché troppo elevate. Si è quindi deciso di eliminare le tuple con questi valori.



Il grafico rappresenta la distanza totale per ogni percorso di ogni classe. Si può notare come ci sono vari percorsi a piedi con distanze troppo elevate che quindi sono da considerarsi sbagliati. I ‘walk’ da eliminare sono 7.

```
> nrow(dati[dati$label == "walk" & dati$vel_avg > 3.2*3.6,])
[1] 7
```

Prima integrazione

Allo scopo di avere una maggiore informazione per ottenere una corretta predizione della label abbiamo utilizzato, in aggiunta a quello già presente, un dataset contenente le informazioni di ‘city’, ‘state’, e ‘country’ di partenza e di arrivo per ogni percorso.

All’inizio è stato scelto il dataset di OpenStreetMap il quale è stato interrogato facendo delle query al server, tuttavia, una volta unito con l’altro dataset, per molti percorsi almeno uno tra i campi ‘city’, ‘state’ e ‘country’ erano impostati a ‘Not Found’. Questo avveniva perché non riusciva ad associare i dati di OSM con le coordinate di inizio e/o di fine percorso. Infatti su 3.081 tuple di 5.514 non è stato in grado di trovare il corretto valore. Si è quindi deciso di cambiare dataset.

```
> print(sum(dati$stateStart == "State Not Found"))
[1] 2847
> print(sum(dati$stateEnd == "State Not Found"))
[1] 2842
> print(sum(dati$countryStart == "Country Not Found"))
[1] 2850
> print(sum(dati$countryEnd == "Country Not Found"))
[1] 2845
> print(sum(dati$cityStart == "City Not Found"))
[1] 3003
> print(sum(dati$cityEnd == "City Not Found"))
[1] 2987
```

Numero di valori ‘Not Found’ per feature.

Il secondo dataset che abbiamo provato ad utilizzare è stato quello di Google Maps. In questo caso abbiamo riscontrato un problema relativo alla quantità di query effettuabili al giorno.
Infatti, il numero di query effettuabili nell'arco delle 24 ore, è limitato a 2500. Quindi, avendo 5514 tuple, e considerando che per ogni tupla sono necessarie due query (una per le coordinate iniziali e una per le coordinate finali) avremmo bisogno di almeno 4 giorni per poter eseguire tutte le query, risultando perciò infattibile usare questo dataset per l'integrazione.

Alla fine si è pensato di usare il dataset di bing, il quale ha il vantaggio di avere un limite di query effettuabili giornalmente di 50.000 (quindi superiore al precedente e non problematico per la necessità minima di query dettata dal nostro problema), oltre ad avere l'accuratezza dei dati maggiore rispetto ad OpenStreetMap.
Infatti delle 5.514 tuple del dataset solo 5 hanno avuto problemi di riconoscimento di 'city', 'state' e 'country', casi per i quali si è reso quindi necessario un processo di integrazione manuale. Grazie a questa fase di integrazione un solo percorso è stato eliminato poiché individuava un punto nel mare. Per i rimanenti sono state aggiunte features necessarie alla successiva fase di machine learning.

	stateStart	countryStart	cityStart	stateEnd	countryEnd	cityEnd
1844	Hainan	China	Qionghai	Hainan	China	City Not Found
1845	Hainan	China	City Not Found	Hainan	China	City Not Found
3617	State Not Found	People's Republic of China	City Not Found	Inner Mongolia	China	Ergun City
3726	Hong Kong	Hong Kong-China	Tsuen Wan	Hong Kong	Hong Kong-China	City Not Found
3916	WA	United States	SeaTac	State Not Found	Country Not Found	City Not Found

Tabella riportante i percorsi critici.

Seconda integrazione

Per effettuare la seconda integrazione abbiamo usato nuovamente OpenStreetMap.
In questo caso vogliamo ottenere il tipo di "via", più vicina ai punti del percorso.
All'inizio si è considerato di prendere tutti gli n punti del percorso ottenuti dalla seconda versione del dataset e vedere per ognuno di essi il tipo di via.
Per trovare la via si considerano tutte le vie in un raggio di 150 metri dalle coordinate del punto e si prende quella la cui distanza è minore. Se non si considerasse il raggio intorno al punto si rischierebbe di associare al punto stesso palazzi o altri edifici. Infatti la precisione dei dati GPS è di qualche metro e quindi non si ha un grado di precisione abbastanza elevato per considerare solo il punto senza un intorno.
Per ottenere i dati è necessario effettuare le query al server OSM. Vengono però chieste solo le mappe con strade, aeroporti e ferrovie.
Tuttavia, usando questo approccio, quando abbiamo effettuato le query al server OSM il tempo di esecuzione per ognuna di esse risultava troppo alto e quindi effettuarne 4.000.000

non si è rivelato fattibile. Inoltre il server dopo un determinato numero di richieste interrompeva la connessione con il nostro calcolatore.
Per superare questo problema abbiamo deciso di considerare solo il punto iniziale e il punto centrale del percorso e per evitare l'interruzione della connessione si è deciso di cambiare server di OpenStreetMap.
Sono state scaricate le mappe per il punto iniziale e centrale di ogni percorso e tramite il nostro script sono state unite in una singola mappa. Si è quindi caricata la mappa così ottenuta su ira_open_street_map (software sviluppato dal laboratorio iraLab) per effettuare le query. Lo script è stato modificato per accettare le mappe di highway, railway e aeroway e per restituire il tag della via.
Vengono poi eseguite le query da R a ira_open_street_map ottenendo la via più vicina al punto, nell'intorno di 50 metri, con il relativo tag che la identifica. Se non viene trovata una via nell'intorno viene ritornato NA.

La query ritorna complessivamente due file csv, il primo per il punto iniziale, il secondo per il punto centrale. Nel file sono presenti tre nuove tag: aeroway, railway e highway. Inoltre il tag highway contiene un altro tag chiamato sidewalk che identifica i marciapiedi a lato della strada. Tuttavia in Cina questo tag non quasi mai usato.

	n_row	Latitude	Longitude	highway	aeroway	railway	sidewalk
1	23	41.14214	80.28686	highway: "primary"	aeroway: "	railway: "	sidewalk: "
2	32	41.16885	80.26397	highway: "tertiary"	aeroway: "	railway: "	sidewalk: "
3	48	39.49383	76.04427	highway: "service"	aeroway: "	railway: "	sidewalk: "
4	54	39.47342	76.00025	highway: "secondary"	aeroway: "	railway: "	sidewalk: "
5	617	39.47418	75.99098	highway: "footway"	aeroway: "	railway: "	sidewalk: "
6	792	39.47476	75.98638	highway: "secondary"	aeroway: "	railway: "	sidewalk: "
7	1058	39.46790	76.01778	highway: "primary"	aeroway: "	railway: "	sidewalk: "
8	1074	41.69335	83.01396	highway: "service"	aeroway: "	railway: "	sidewalk: "
9	1083	41.70710	82.97737	highway: "trunk"	aeroway: "	railway: "	sidewalk: "
10	1094	41.71068	82.96478	*NA*	*NA*	*NA*	*NA*
11	1478	41.70673	82.99532	highway: "trunk"	aeroway: "	railway: "	sidewalk: "
12	37672	42.80188	86.37128	highway: "	aeroway: "	railway: "rail"	sidewalk: "

Esempio di possibili valori dei tre nuovi tag

Numero di valori NA per colonna

```
> print(colSums(is.na(dati_start)))
   n_row  Latitude Longitude   highway   aeroway   railway  sidewalk
       0         0         0        93        93        93        93
> print(colSums(is.na(dati_middle)))
   n_row  Latitude Longitude   highway   aeroway   railway  sidewalk
       0         0         0       268       268       268       268
```

Viene quindi ora considerato il punto iniziale. Se è presente un tipo di via in uno dei tre tag allora viene considerato quello come il valore per tutto il percorso.

Il tag highway ha diversi tipi. Considero i tipi footway, pedestrian, runway e steps come un solo insieme chiamato pedestrian (come documentazione di OSM). Il tipo cycleway non cambia il proprio nome, mentre tutti gli altri tipi sono raggruppati nell'insieme road.
Se invece non è presente un tipo di via in uno dei tre tag del punto iniziale (ovvero se ho NA), si considerano i tag del punto centrale, mentre se ci sono tuple senza valori dei tag si mette di default il valore pedestrian perché si considera il caso in cui l'utente stia facendo una camminata in un luogo dove non ci sono strade (es. montagna). I percorsi senza tipo di via sono 22.

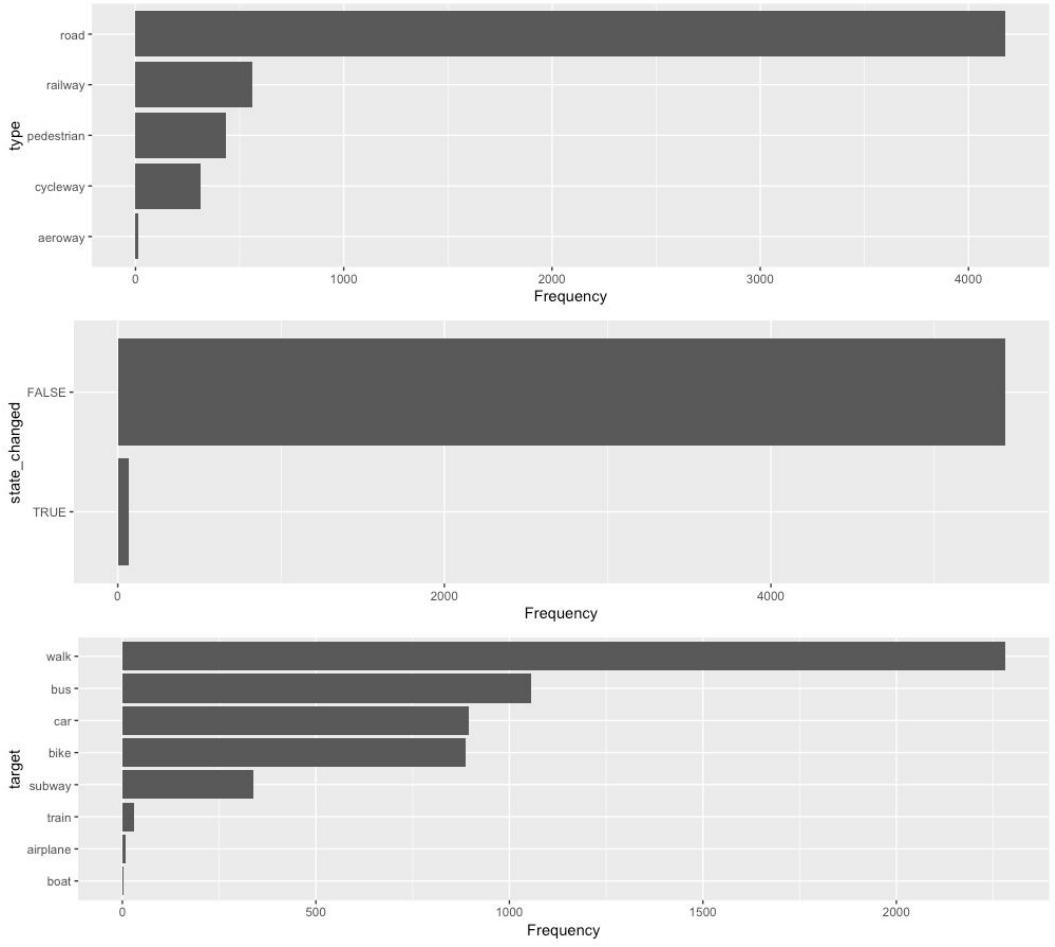
```
> # valori che non hanno tag che mettiamo a pedestrian
> print(n_val_null)
[1] 22
```

Il dataset finale non presenta valori NA ed è completo, presenta 5514 tuple con 27 features.

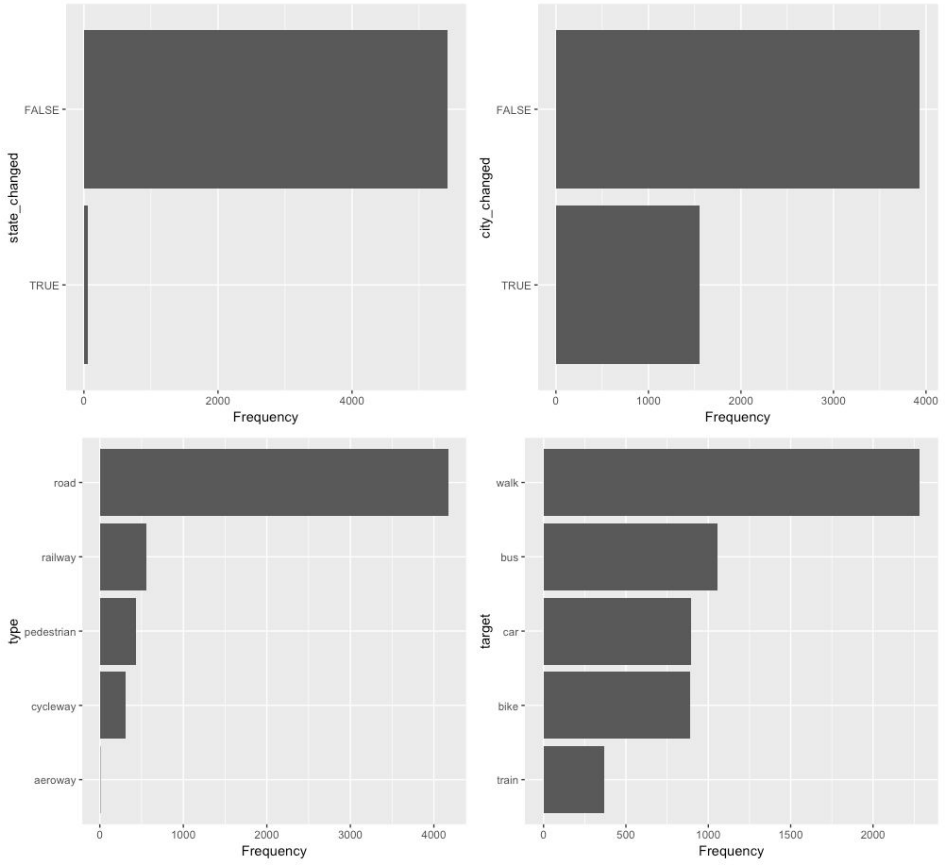
Analisi descrittive dei dati

In questa sezione si analizzano alcuni aspetti del dataset.

Analisi sulle classi



Il primo grafico rappresenta la distribuzione di frequenza per tipo di via. Si può notare come ‘road’ è il tipo di tipo di via più frequente tra i vari percorsi, seguito da railway, pedestrian, cycleway e infine aeroway.
Il secondo grafico rappresenta la distribuzione di frequenza per i percorsi il cui stato finale è uguale o diverso da quello iniziale. Si può notare come nella maggior parte dei percorsi lo stato finale coincida con quello iniziale.
Osservando quindi l’ultimo grafico in unione agli altri, possiamo notare come le classi boat e airplane sono meno rappresentate nel dataset rispetto alle altre classi. Perciò si è deciso di eliminarle dal dataset perchè non è possibile effettuare una generalizzazione corretta per queste classi essendoci meno tuple con queste label.
Inoltre a Pechino, dove è stata registrata la maggior parte delle 5.514 tuple, subway e train sono perfettamente connesse, infatti si considera come subway anche la metropolitana all’aperto e quindi si possono fondere insieme in un nuova classe train.
Si ha quindi un nuovo grafico (in basso a destra), in cui frequenza delle classi è più omogenea rispetto a quello precedente.
I grafici in alto a sinistra e in basso a sinistra rappresentano le stesse informazioni dei grafici visti in precedenza. Invece il grafico in alto a destra rappresenta la frequenza con cui si cambia di città quando si effettua un percorso. Si può notare come per la maggior parte dei percorsi non si cambia città tra il punto iniziale e il punto finale.



Machine learning

Il dataset derivante dalla fase precedente di Data Integration presenta 5.513 tuple e ha 27 attributi:

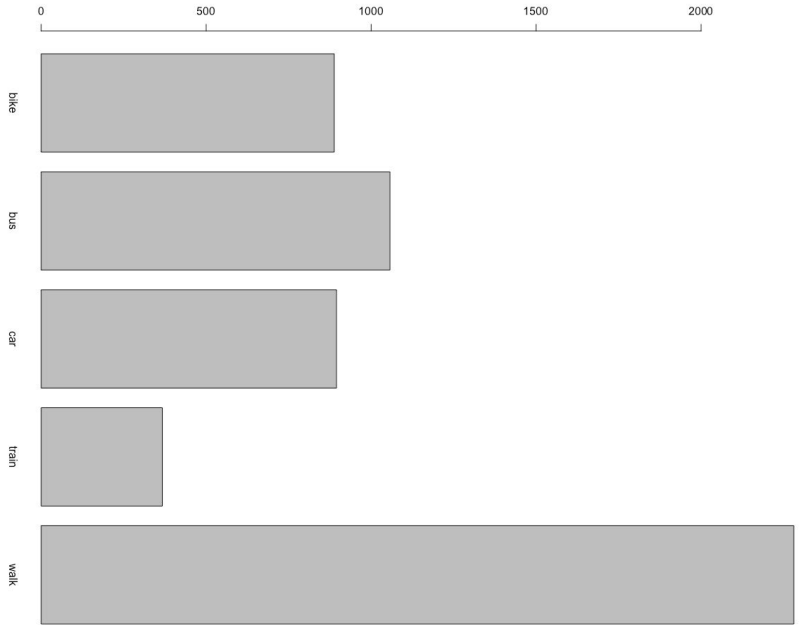
Id_user, Id_perc labe, longitudeStart, latitudeStart, latitudeEnd, longitudeEnd, TimeStart, TimeEnd, distanceTotal, time_total, n777, npoints, vel_avg, vel_max, altitudeAvg, altitudeMax, vcr, sr, hcr, stateStart, countryStart, cityStart, stateEnd, countryEnd, cityEnd, tag

Essendo poco significativo lo stato di partenza e di arrivo abbiamo deciso trasformare in una variabile booleana il fatto del cambio di stato dalla partenza del percorso all'arrivo dello stesso, la stessa cosa è stata fatta per le city.

Eliminiamo le tuple con label ad "Airplane" perché sono solo 8 tuple e sarebbero poco significative per l'analisi.
Eliminiamo le tuple con label ad "Boat" perché sono solo 4 tuple e sarebbero poco significative per l'analisi.

Avendo la label "car" e la label "taxi" le abbiamo uniformate in un'unica classe chiamata "car", la label "run" essendo composta da sole 3 tuple l'abbiamo incorporata con la label "walk" .
Inoltre a Pechino, dove è stata registrata la maggior parte delle 5.513 tuple, subway e train sono perfettamente connesse, infatti si considera come subway anche la metropolitana all'aperto e quindi si possono fondere insieme in un nuova classe train.

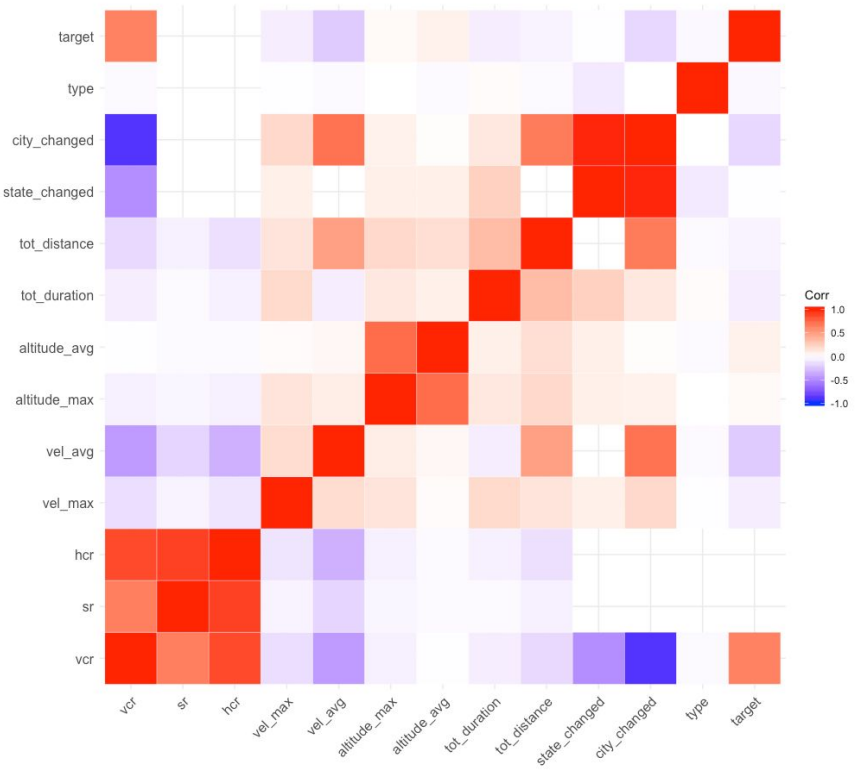
Le classi che abbiamo sono così distribuite:



Analisi esplorativa dei dati

La matrice di correlazione misura il grado di correlazione lineare per ogni coppia di feature e viene visualizzata nel seguente modo:

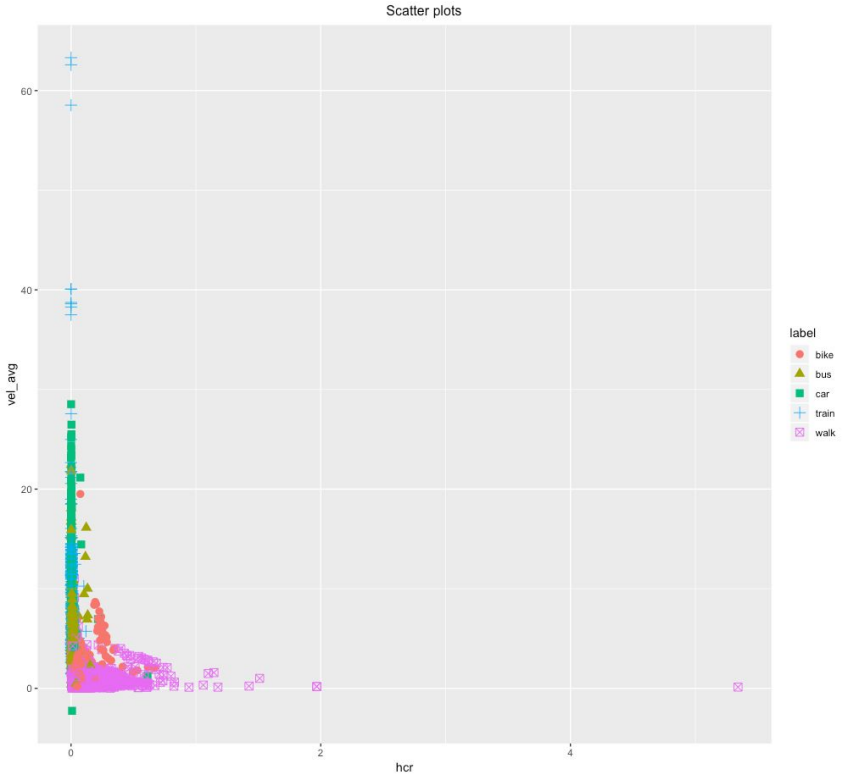
- **Correlazione direttamente proporzionale**: con dei quadrati rossi, corrispondenti a valori (0,1]
- **Correlazione inversamente proporzionale**: con dei quadrati blue, corrispondenti a valori (0,-1]



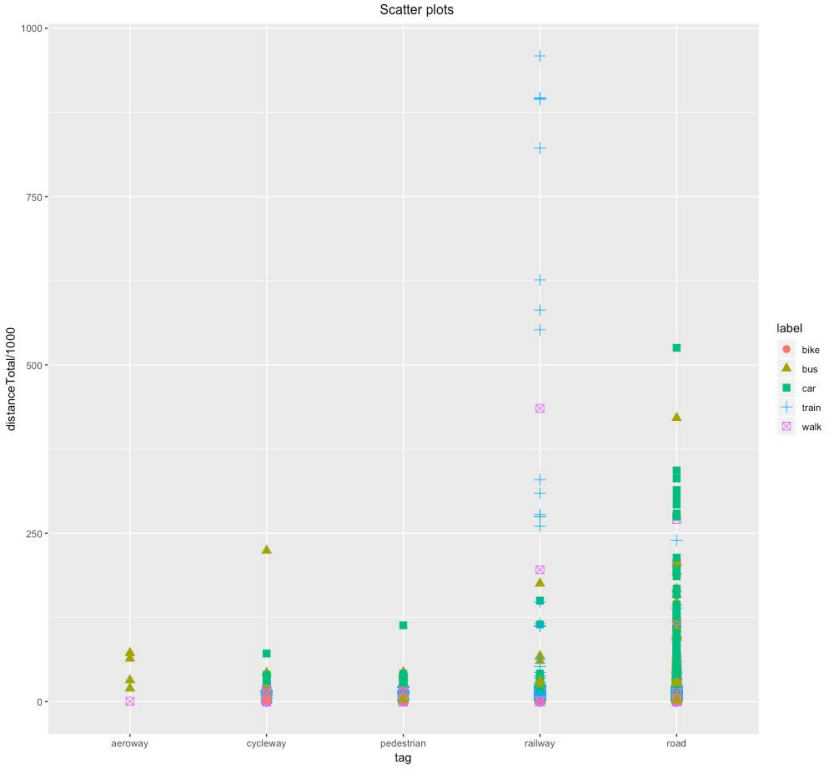
Nello specifico, tramite questa matrice, è possibile comprendere che:

- **vcr** e **city_changed** sono inversamente correlate (blu intenso = -1.0), quindi questo ci fa capire che quando si cambia città la velocità non varia molto.
- **altitude_avg** e **altitude_max** sono abbastanza correlate (circa 0.8), quindi questo ci fa capire che normalmente l'altitudine media e massima hanno valori vicini tra di loro.
- **city_changed** e **vel_avg** sono abbastanza correlate (circa 0.8), quindi questo ci fa capire che cambiando città la velocità media dei percorsi non varia molto.
- **city_changed** e **tot_distance** sono abbastanza correlate (circa 0.8), quindi questo ci fa capire che quando cambia la città la distanza totale è elevata.
- **hcv sr** e **vcr** sono molto correlati tra di loro
- **vel_avg** e **city_changed** sono correlati fra loro
- **vel_avg** e **hcr** sono inversamente correlati poiché in città avrò un alto cvr ma una bassa velocità media.

Come già visto dalla matrice di correlazione si può notare la divisione delle label in relazione alla velocità media e del hcr, infatti le tuple con label "walk" hanno un hcr alto e una velocità media bassa, invece le tuple con label "car" o "train" hanno una velocità media alta e un hcr basso



Dallo scatter plot che raffigura le label come sono distribuite del dataset in funzione del tag di open street map e della sistanza del percorso possiamo vedere come i tag di open street map riescono a identificare abbastanza bene le varie classi delle label.



Esperimenti condotti

Dopo aver preparato il dataset con le features ritenute necessarie alla classificazione abbiamo condotto due esperimenti:

- Classificazione tramite modello *SVM* (support vector machine)
- Classificazione tramite modello *Naive Bayes*

Divisione del dataset

Al fine di effettuare il task di classificazione, il dataset è stato suddiviso in due parti:

- Training_set : 70%
- Test_set : 30%

Per ognuno dei modelli utilizzati è stata effettuata una k-fold cross validation.
In particolare è stata ripetuta per dieci volte su dieci fold.
Al fine di applicare questa tecnica di validazione abbiamo utilizzato la funzione "*trainControl*" fornita dal package "*caret*", specificandone i parametri "method = "repeatedcv", number = 10, repeats = 10".
Siccome le classi del nostro dataset non sono perfettamente bilanciate, provando ad eseguire manualmente la k-fold cross validation, capitava spesso che in uno dei fold una classe non fosse per nulla rappresentata generando così un errore in fase di costruzione della matrice di confusione (non essendo il numero di classi nel training e nel validation di ugual numero).

SVM

Le Support Vector Machine sono dei modelli di apprendimento supervisionato associati ad algoritmi di apprendimento per la regressione ed in particolare, utile nel nostro caso, per la classificazione.
Il nostro scopo era predirre, sulla base di una serie di percorsi registrati principalmente nella città di Pechino, il mezzo di trasporto utilizzato negli stessi.
Abbiamo utilizzato la funzione "*svm*" del package "*e1071*".

Fase di training

Il modello è stato addestrato utilizzando una Support Vector Machine con kernel di tipo "*radial".*
Il parametro di costo è stato posto a dieci dopo una serie di esperimenti.
Sul training_set viene eseguita la 10 fold cross validation con un sampling proporzionato per ogni fold. Questa tecnica è utile per evitare che il modello vada in *overfitting.*
Abbiamo utilizzato un kernel di tipo "*radial*" perchè avendo otto classi era difficile approssimare la separazione delle stesse con una funzione lineare.

Fase di test

Addestrato il modello sul training set, è stato possibile effettuare la predizione delle classi sul test_set (che non era ancora stato toccato dopo il primario split del dataset).

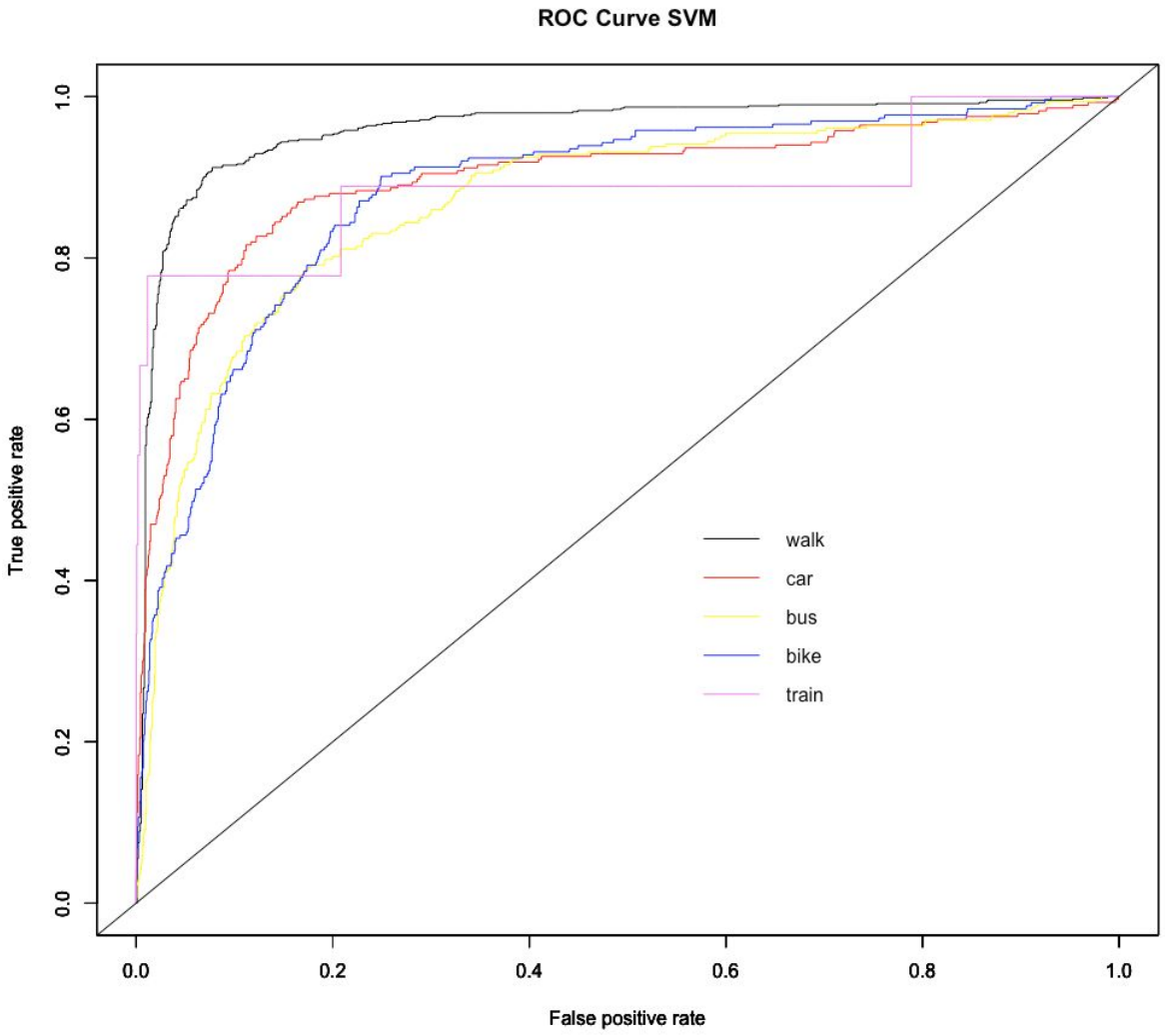
Matrice di confusione

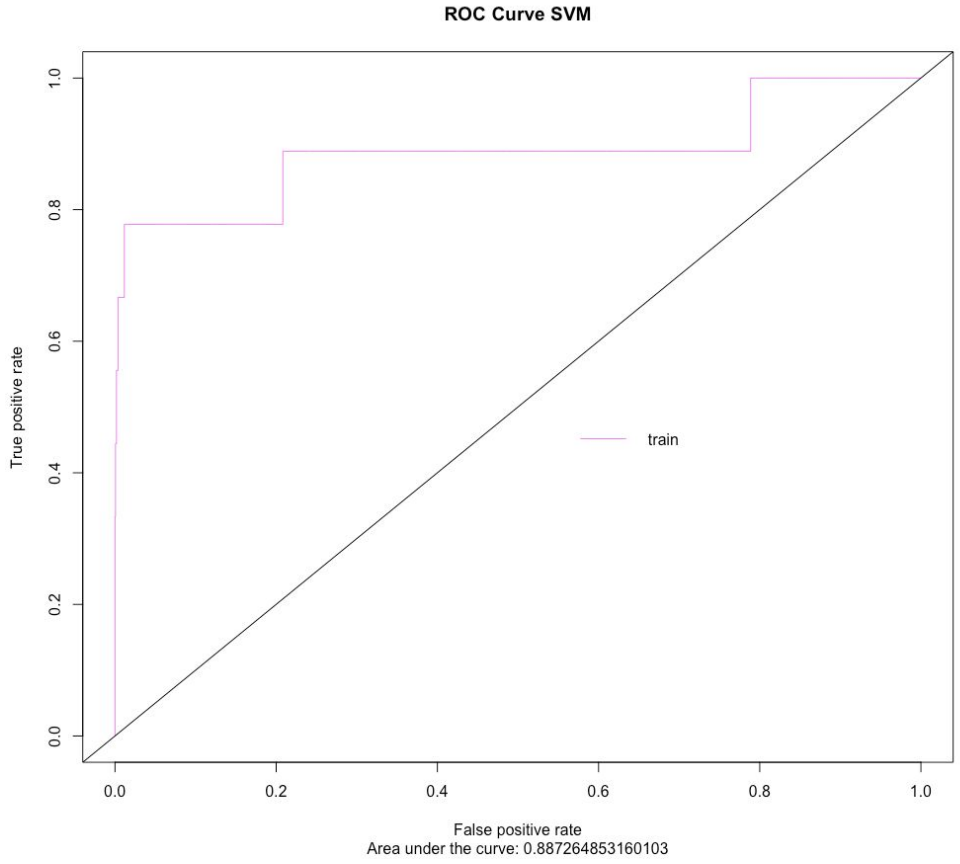
```
Confusion Matrix and Statistics

          Reference
Prediction bike bus car train walk
     bike   222  39   8     0   37
     bus     14 211  56    23   17
     car      0  30 173    31    5
     train    0   9  10    37    2
     walk    35  15  12     2  657

Overall Statistics

               Accuracy : 0.7903
                 95% CI : (0.7698, 0.8097)
    No Information Rate : 0.4365
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.7075
 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: bike Class: bus Class: car Class: train Class: walk
Sensitivity               0.8192     0.6941     0.6680      0.39785      0.9150
Specificity               0.9389     0.9180     0.9524      0.98647      0.9310
Pos Pred Value            0.7255     0.6573     0.7238      0.63793      0.9112
Neg Pred Value            0.9634     0.9298     0.9388      0.96471      0.9340
Prevalence                0.1647     0.1848     0.1574      0.05653      0.4365
Detection Rate            0.1350     0.1283     0.1052      0.02249      0.3994
Detection Prevalence      0.1860     0.1951     0.1453      0.03526      0.4383
Balanced Accuracy         0.8790     0.8060     0.8102      0.69216      0.9230
```

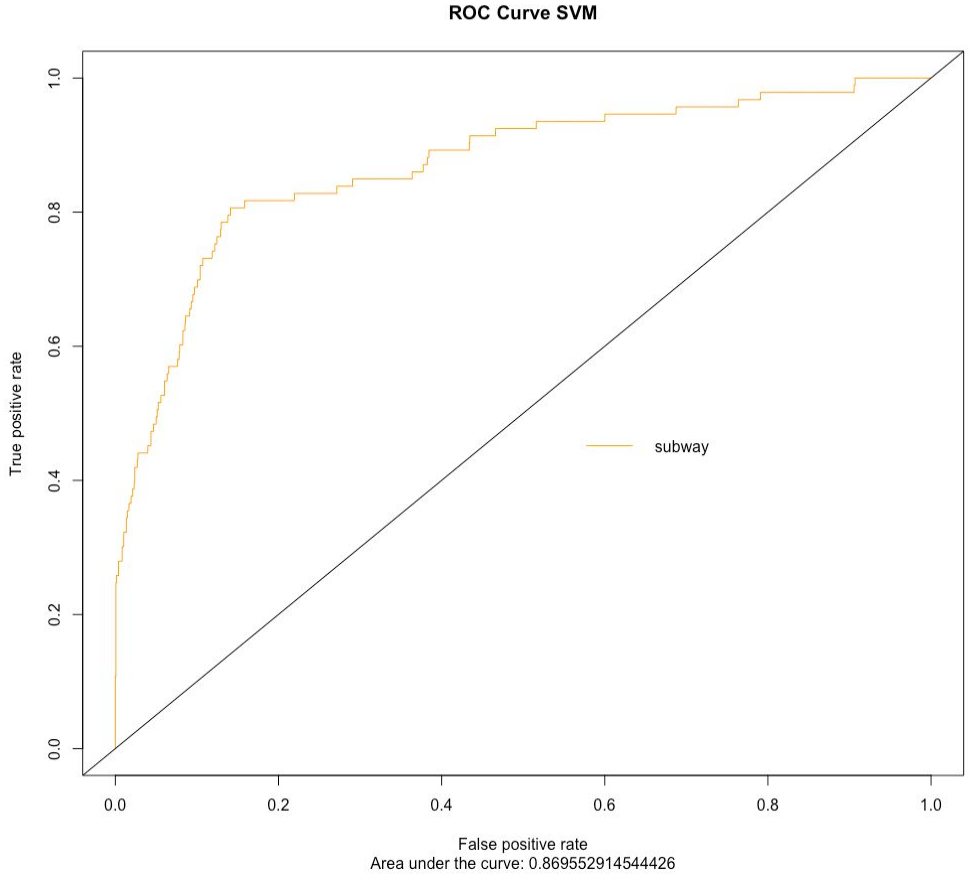
Come si può notare dalla matrice di confusione le classi non sono perfettamente bilanciate. Nonostante ciò, sul test_set il modello ottiene il 79% accuracy.
Utilizzando lo stesso modello, è stata analizzata la curva ROC.
Essendo il nostro problema multiclasse ed essendo la ROC per sua natura caratterizzante problemi di tipo binario, è stato necessario effettuare una binarizzazione del problema.

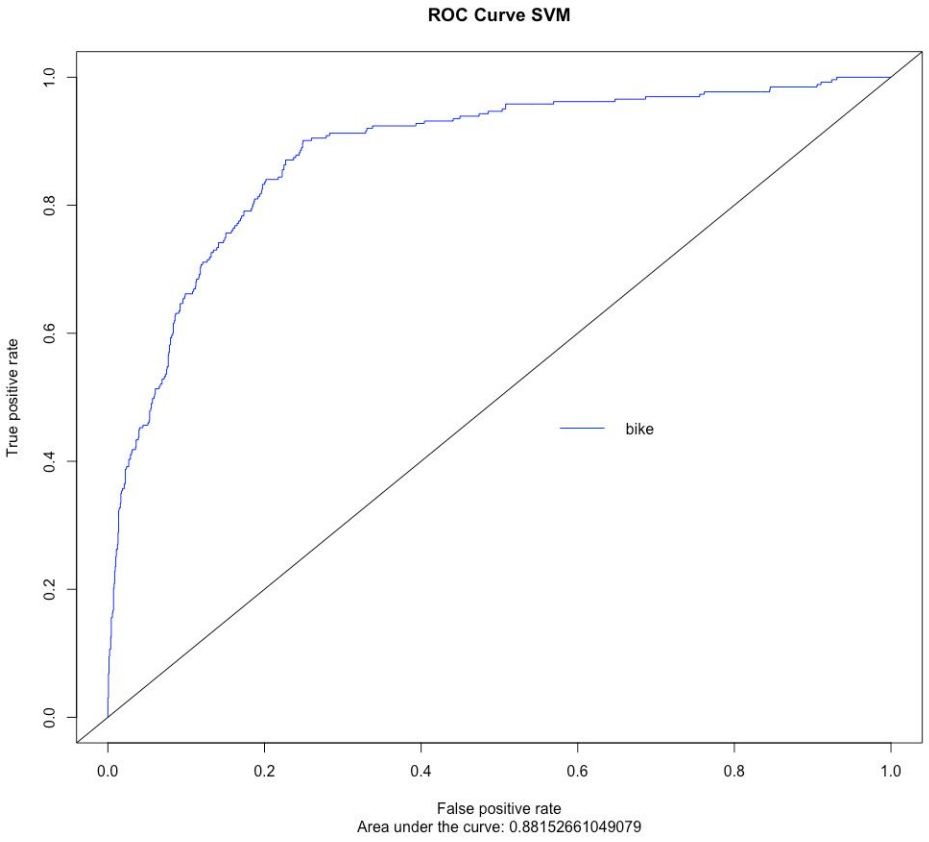
One vs All ROC evaluation

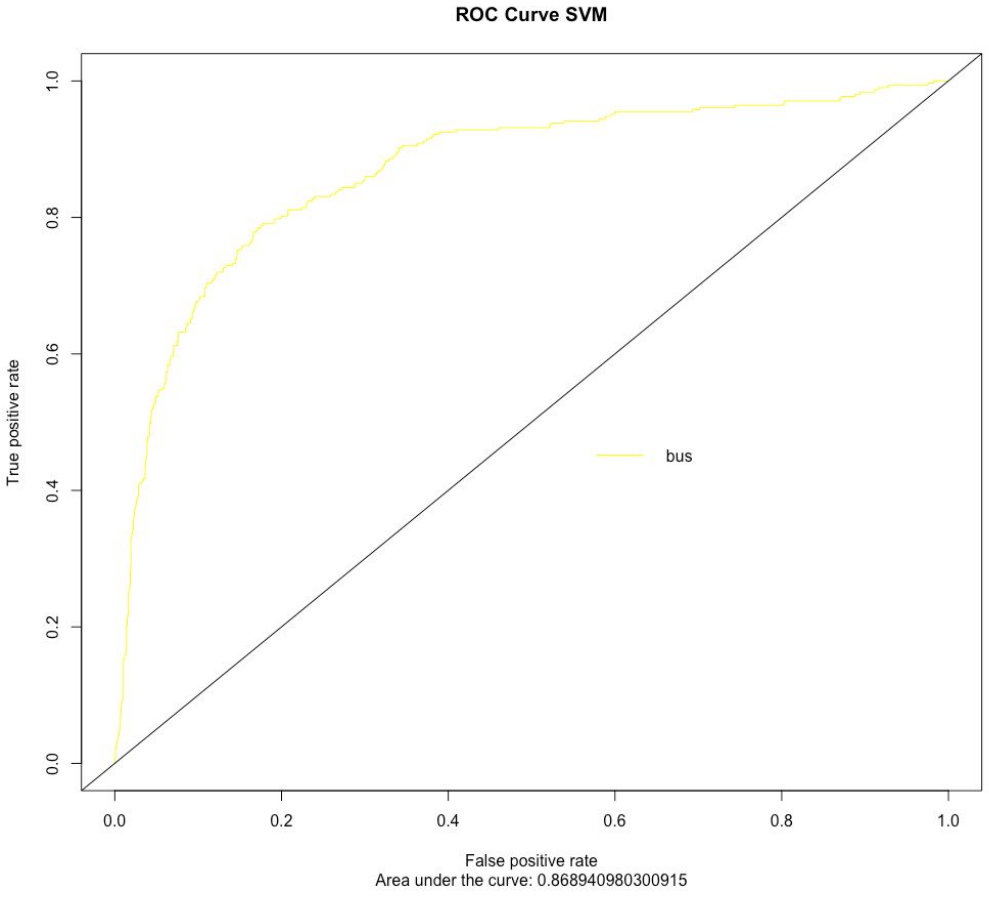
Lungo i due assi si possono rappresentare la sensibilità e (1-specificità), come True Positive Rate (TPR, frazione di veri positivi) e False Positive Rate (FPR, frazione di falsi positivi).
In altre parole rappresenta il rapporto tra hit rate (true positive) e falsi positivi.
Di seguito vengono riportate le curve ROC per ogni classe.

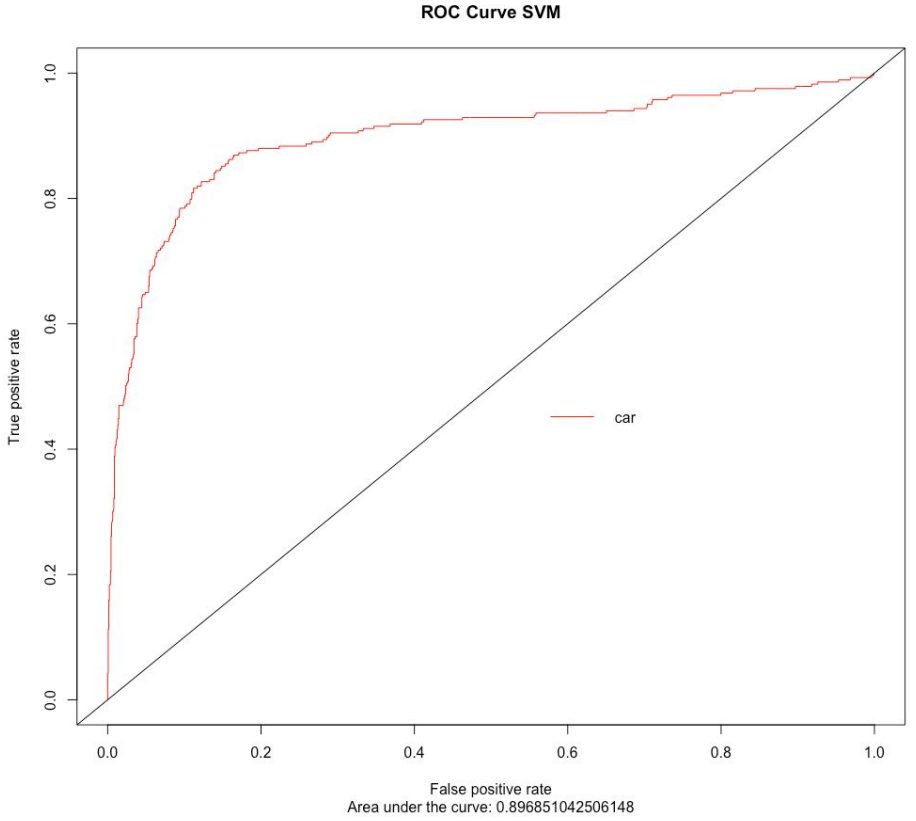

ROC Curve SVM
True positive rate
False positive rate
0.0
0.2
0.4
0.6
0.8
1.0
walk
car
bus
bike
train

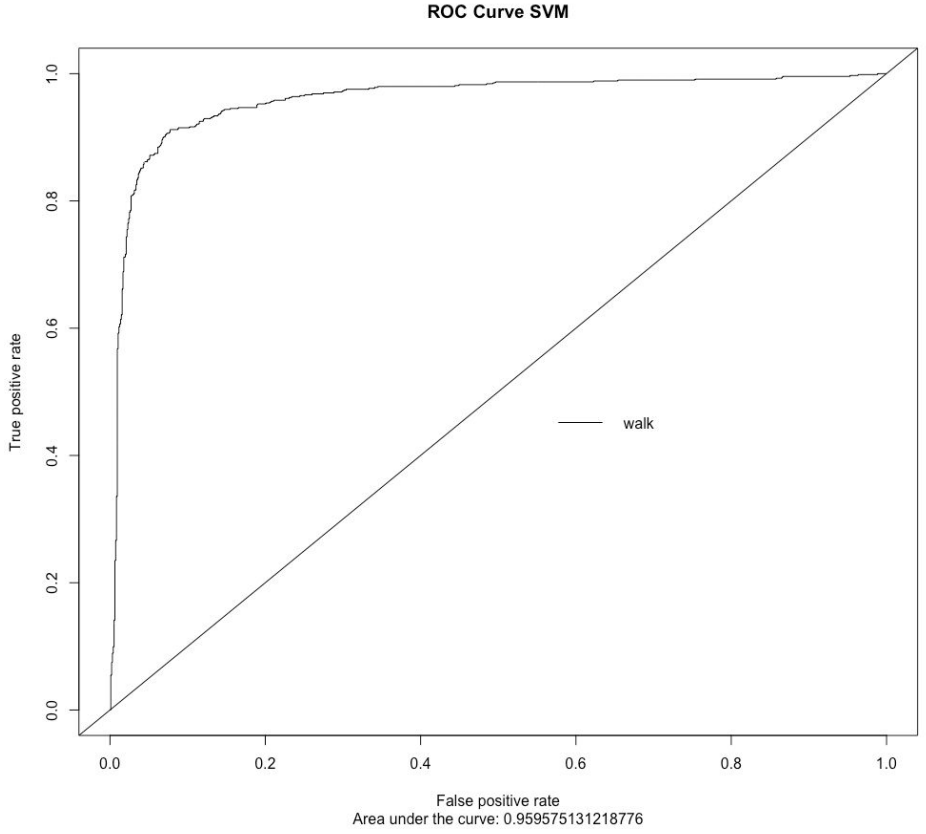

ROC Curve SVM
True positive rate
False positive rate
Area under the curve: 0.887264853160103
train
0.0
0.2
0.4
0.6
0.8
1.0


ROC Curve SVM
True positive rate
False positive rate
Area under the curve: 0.869552914544426
subway
0.0
0.2
0.4
0.6
0.8
1.0


ROC Curve SVM
True positive rate
False positive rate
Area under the curve: 0.88152661049079
bike
0.0
0.2
0.4
0.6
0.8
1.0


ROC Curve SVM
True positive rate
False positive rate
Area under the curve: 0.868940980300915
bus
0.0
0.2
0.4
0.6
0.8
1.0


ROC Curve SVM
True positive rate
False positive rate
Area under the curve: 0.896851042506148
car
0.0
0.2
0.4
0.6
0.8
1.0


ROC Curve SVM
True positive rate
False positive rate
Area under the curve: 0.959575131218776
walk
0.0
0.2
0.4
0.6
0.8
1.0

La media totale delle Area Under Curve è di 0.910299. Buon indicatore della bontà del modello.

METRICHE SVM PER CLASSE

```
> precision
     bike       bus       car     train      walk
0.6710963 0.6055046 0.7402135 0.6521739 0.8739130
> recall
     bike       bus       car     train      walk
0.7318841 0.6556291 0.7098976 0.2654867 0.9122542
> f1
     bike       bus       car     train      walk
0.7001733 0.6295707 0.7247387 0.3773585 0.8926721
```

Dove con f1 intendiamo *"f-measure"*

NAIVE BAYES

Abbiamo utilizzato la funzione "*train*" del pacchetto "*caret*" per addestrare questo modello.
E' stata effettuata una ten cross validation ripetuta per dieci volte con le stesse modalità illustrate in precedenza tramite la funzione "trainControl".

Fase di training

Il training è stato eseguito sullo stesso training set del precedente modello per poter effettuare un confronto obiettivo.
Specificando il parametro *method = 'nb'* nella funzione train è stato possibile addestrare il classificatore.

Fase di test

La fase di test è stata eseguita sullo stesso training set utilizzato per il precedente modello con le stesse features.

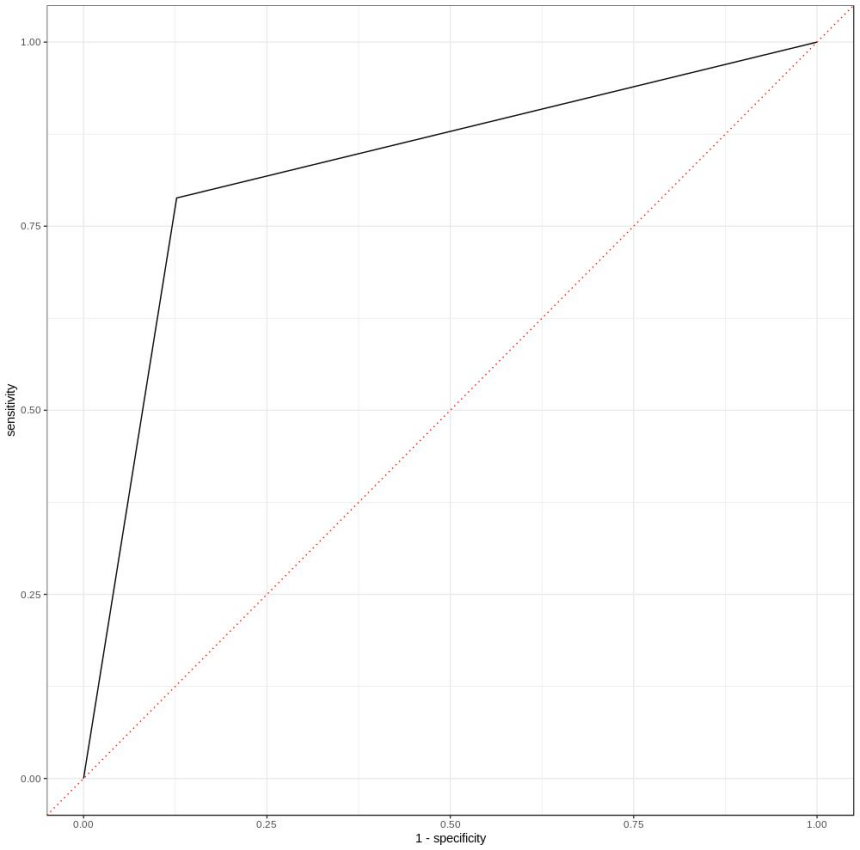
```
Confusion Matrix and Statistics

          Reference
Prediction bike bus car train walk
     bike   209 102  22     2   92
     bus      4 144  48    14   21
     car      1  37 180    72    9
     train    0   1   2    36    3
     walk    24  11   1     5  605

Overall Statistics

               Accuracy : 0.7137
                 95% CI : (0.6912, 0.7354)
    No Information Rate : 0.4438
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.6094
 Mcnemar's Test P-Value : < 2.2e-16

Statistics by Class:

                     Class: bike Class: bus Class: car Class: train Class: walk
Sensitivity               0.8782    0.48814     0.7115      0.27907      0.8288
Specificity               0.8451    0.93556     0.9145      0.99604      0.9552
Pos Pred Value            0.4895    0.62338     0.6020      0.85714      0.9365
Neg Pred Value            0.9762    0.89321     0.9458      0.94198      0.8749
Prevalence                0.1447    0.17933     0.1538      0.07842      0.4438
Detection Rate            0.1271    0.08754     0.1094      0.02188      0.3678
Detection Prevalence      0.2596    0.14043     0.1818      0.02553      0.3927
Balanced Accuracy         0.8616    0.71185     0.8130      0.63756      0.8920
```

Come si può vedere dalle statistiche ottenute nella fase di test, tramite il modello Naive Bayes otteniamo una minore accuracy totale sul test_set.
Ma classe per classe ci sono delle differenze; ad esempio la classe train viene riconosciuta più volte correttamente utilizzando il modello bayesiano piuttosto che le macchine a vettori di supporto.

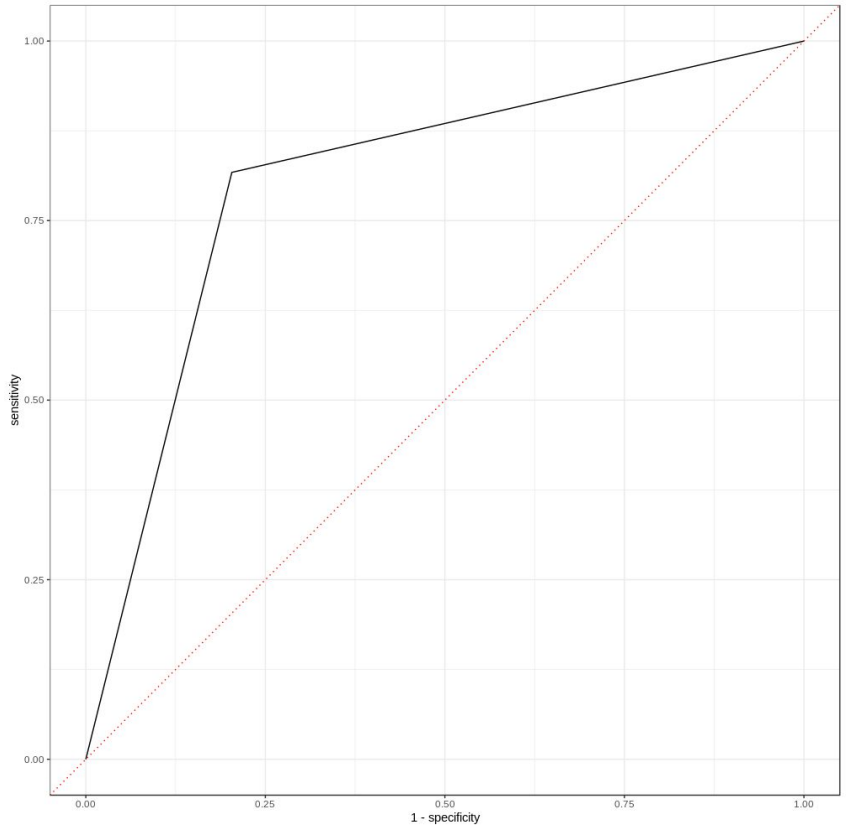
One vs All ROC evaluation

Di seguito sono riportate le curve ROC del modello Naive Bayes.

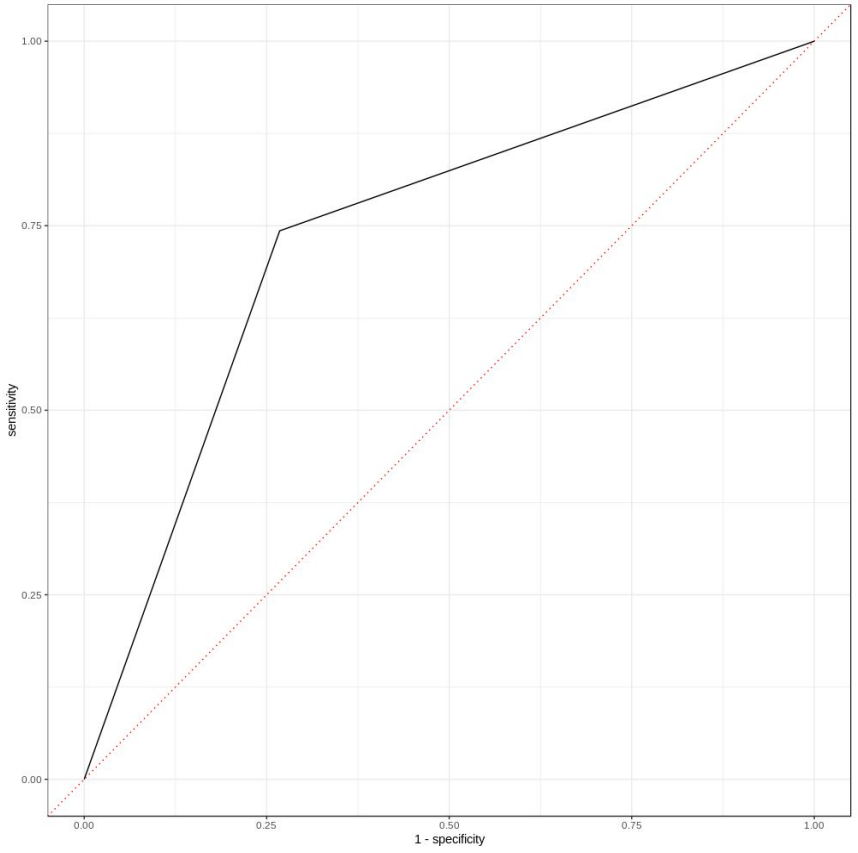
Roc della classe car con AUC = 0.8307



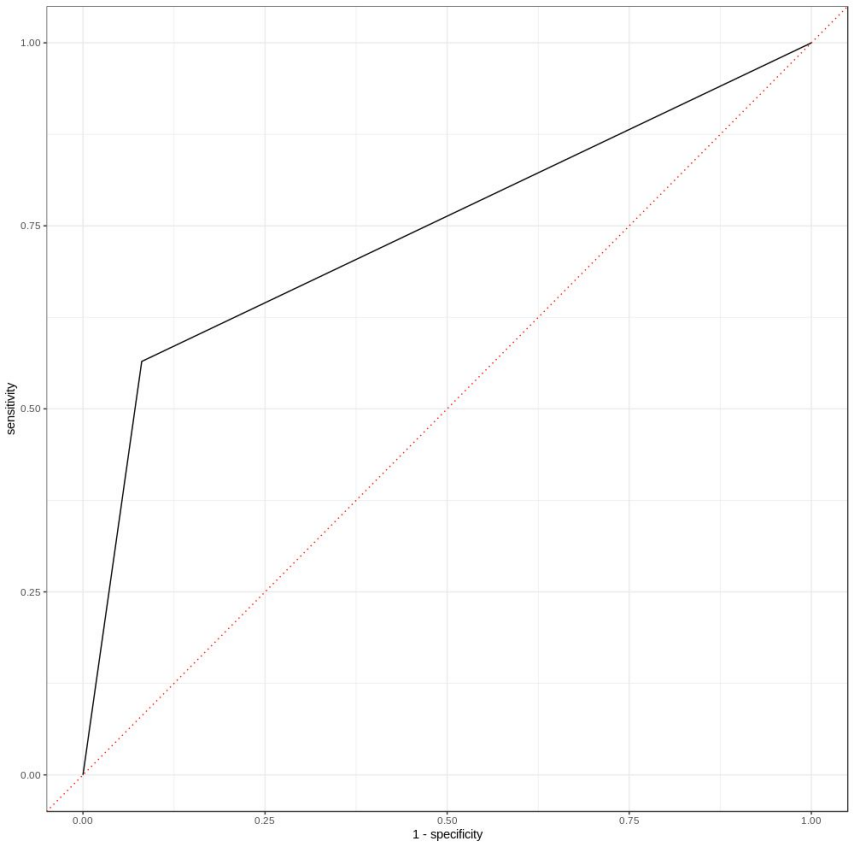
Roc della classe bike con AUC = 0.8069



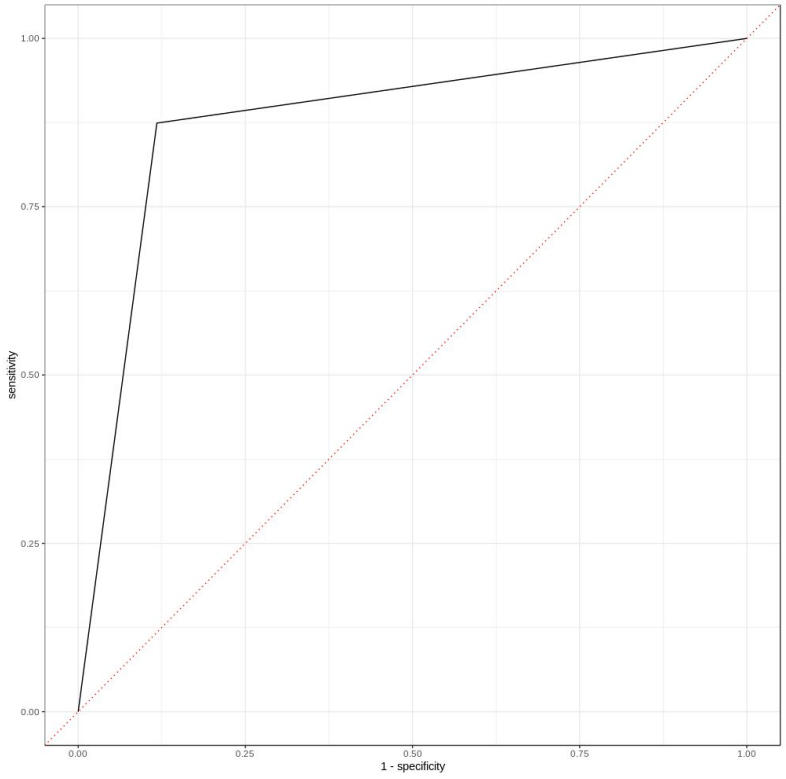
Roc della classe bus con AUC = 0.7376



Roc della classe train con AUC = 0.7421



Roc della classe di walk con AUC = 0.8783



La media totale delle Area Under Curve è di 0.79912. indicatore della minore bontà del modello rispetto al precedente.

METRICHE NAIVE BAYES PER CLASSE

```
> accuracy
[1] 0.7136778
> precision
     bike       bus       car     train      walk
0.4894614 0.6233766 0.6020067 0.8571429 0.9365325
> recall
     bike       bus       car     train      walk
0.8781513 0.4881356 0.7114625 0.2790698 0.8287671
> f1
     bike       bus       car     train      walk
0.6285714 0.5475285 0.6521739 0.4210526 0.8793605
```

Dove con f1 intendiamo *"f-measure"*

Conclusioni

Per poter effettuare una previsione più accurata della tipologia di mezzo è necessario considerare anche le classi airplane e boat che sono state scartate durante la creazione del dataset perché il loro numero di tuple era decisamente inferiore alle altre classi.
Bisognerebbe quindi aggiungere nel dataset un quantitativo sufficiente di tuple (ovvero di percorsi) affinché il loro numero non si discosti troppo dal numero di tuple delle altre classi.
Considerando i modelli di classificazione implementati, abbiamo riscontrato una maggiore accuracy con il modello svm, mentre naive bayes ha ottenuto un' accuracy inferiore.
Tenendo conto che vengono usati gli stessi training set e test set per entrambi i modelli, siamo giunti alla conclusione che svm si adatti meglio al nostro problema di classificazione.