# Analisi del Dataset: Fatal Police Shootings in the US

Davide A. Zollo, Matricola: 486136
Luca Carloni, Matricola: 500138

Luglio 2020

## 1 Introduzione

Il nostro progetto finale del corso di Big Data 2019/2020 prevede l'implementazione di un sistema che sia in grado di processare ed analizzare i dati provenienti da diversi CSV i quali nel loro insieme andranno a descrivere sparatorie mortali che coinvolgono la polizia negli Stati Uniti dal 2015 ad oggi. Le tecnologie utilizzate per realizzare la parte di analisi e processing ed infine di visualizzazione dei dati sono le seguenti:

- *Batch Analysis con Apache Spark*
- *Deploy su EC2 di Amazon Web Services*
- *Classificazione con Apache Spark MLlib*
- *Visualizzazione con d3.js*

L'implementazione è stata testata sia in locale che su cluster, i dataset utilizzati per il testing del sistema implementato sono:

1. *MedianHouseHoldIncome2015*, contiene dati del censimento degli Stati Uniti contenenti il reddito familiare medio in tutte le città degli Stati Uniti.
2. *PercentOver25CompletedHighSchool*, contiene dati del censimento degli Stati Uniti che mostrano il tasso (in percentuale) relativo alle persone con più di 25 anni che hanno conseguito il diploma di scuola superiore. Questi dati sono relativi a tutte le città degli Stati Uniti.
3. *PercentagePeopleBelowPovertyLevel*, contiene dati molto simili alla prima tabella descritta con l'eccezione però che esso indica il livello medio di povertà per città.
4. *PoliceKillingUS*, un dataset che contiene i dati registrati dalle sparatorie come il nome del soggetto ucciso con la data, il sesso, la razza, l'età e in che città/stato è avvenuto il tutto, a dati su come è stato ucciso, etc.
5. *ShareRaceByCity*, dataset contenente i dati del censimento degli Stati Uniti riguardanti la demografia etnica in tutte le città degli US.

Il resto della relazione è organizzata come segue: nella prossima sezione vi sarà una descrizione delle tecnologie utilizzate con delle spiegazioni sul perché sono state scelte, nella sezione dopo vengono discusse le analisi effettuate nel dettaglio, partendo dal data cleaning del dataset fino al modello di classificazione realizzato. Infine tutti i dati ottenuti dalle analisi precedenti verranno visualizzati con un apposita tecnologia menzionata in precedenza, ovvero d3.js.

Nella penultima sezione si analizzeranno i risultati ottenuti con annesse una serie di immagini proprio per la visualizzazione di quest'ultimi. Per ultimo invece sarà realizzato un confronto dei tempi di esecuzione ottenuti in locale e nei cluster utilizzati.

# Contenuti

**1 Introduzione** **1**

**2 Tecnologie Utilizzate** **3**

**3 Analisi Effettuate** **4**
3.1 Analisi del Dataset e Data Cleaning . . . . . 4
3.2 Task di base . . . . . 6
3.2.1 Poorest States . . . . . 6
3.2.2 Most Common Victim Name . . . . . 6
3.2.3 State mean High School graduates . . . . . 6
3.2.4 Killed people by state . . . . . 7
3.2.5 Education VS Poverty . . . . . 7
3.2.6 Killed people by race . . . . . 7
3.2.7 Calculate victims to population proportion . . . . . 8
3.2.8 Common manner of death . . . . . 8
3.2.9 Dangerous Cities . . . . . 9
3.2.10 Gender of victim . . . . . 9
3.2.11 Mean age by race . . . . . 9
3.2.12 Most Common Weapon Used . . . . . 10
3.2.13 Body camera Check . . . . . 10
3.2.14 Share race by state . . . . . 10
3.2.15 Killed by month . . . . . 11
3.3 Modello di Classificazione di Machine Learning . . . . . 12
3.3.1 Feature Selection . . . . . 12
3.3.2 Classificatore Multiclasse . . . . . 14
3.3.3 Risultati . . . . . 15

**4 Risultati e Visualizzazione** **17**

**5 Tempi di esecuzione in locale e su cluster** **20**

**6 Conclusioni** **22**

## 2 Tecnologie Utilizzate

Tra le tecnologie utilizzate abbiamo Apache Spark. Spark è un progetto alternativo a MapReduce che ha come obiettivo quello di superare i limiti di quest'ultimo. Per rispondere alla domanda sul perché si è scelto di utilizzare Spark invece di MapReduce basta analizzare nel dettaglio i due sistemi: il problema principale di MapReduce consiste nel fatto che letture e scritture avvengono su disco, quindi vi è un alto costo per le comunicazioni ed è inoltre necessaria la trasmissione di una grande mole di dati sulla rete. Per questo motivo, MapReduce risulta essere inefficiente per **Algoritmi Iterativi** (Machine learning, Graphs e Network Analysis). Dall'altro lato l'idea di base di Spark è quella di eseguire operazioni di I/O in memoria, per cui si sostituiscono ai dischi le SSD.

A seguire per poter andare a effettuare delle analisi più dettagliate si è utilizzato Apache Spark MLlib, una libreria scalabile di Spark contenente algoritmi di machine learning, utilizzabile con linguaggi di programmazione come *Scala*, *Java*, *Python*. MLlib entra nelle API di Spark e interagisce con Numpy in Python e con le librerie di R. Come detto tale libreria contiene algoritmi di Machine Learning per svolgere analisi anche più complesse. I primi algoritmi che possiamo analizzare sono quelli contenuti nella categoria denominata **3C**, contenente algoritmi come il *Collaborative Filtering*, esso rappresenta un set di tecniche per poter effettuare raccomandazione automatica. L'obiettivo di tale algoritmo è quello di predire l'interesse di un utente verso un item oppure quello di filtrare solamente gli item di interesse. Un esempio di applicazione è nell'utilizzo dell'*Alternating least squares (ALS)* mentre un altro algoritmo che rientra anch'esso in questa categoria è quello del **Clustering**, che ha come obiettivo quello di andare a raggruppare oggetti simili. Tale azione può essere intrapresa tramite il calcolo delle distanze tra i vari oggetti, possiamo passare dalla distanza di Manhattan a quella Euclidea. Tra gli esempi di algoritmi, quelli più utilizzati sono il K-Means, Gaussian mixtures. Ultimo algoritmo all'interno della categoria 3C è quello della *Classification*, che analizzeremo più nel dettaglio nelle sezioni successive poiché è stato impiegato nei nostri casi d'uso, tale algoritmo ha come obiettivo quello di porre gli item in delle categorie predefinite. Gli approcci con i quali possiamo intraprendere l'utilizzo di questo algoritmo passano da quello lineare, all'albero di decisione, alle reti Bayesiane. Come vederemo successivamente nel nostro caso specifico siamo andati ad utilizzare una classificazione lineare, più nel dettaglio una SVM multiclasse. Altre categorie in cui possono ricadere gli algoritmi di Machine Learning sono quelle dell'**FPM**, o del **Miscellaneous**. Con il primo si potrebbe intraprendere la ricerca di un set di item più frequente, ad esempio all'interno del contesto alimentare, mentre con il secondo si possono applicare tecniche di regressione come linear regression, logistic regression, etc. Oppure alberi di decisione o approcci Random Forest.

Come detto nell'introduzione l'implementazione è stata distribuita sia in locale che su cluster. Nel dettaglio per l'implementazione su cluster è stato utlizzato AWS, più precisamente Elastic Map Reduce (EMR). Possiamo andare a fare una panoramica dei vantaggi che AWS può offrire, AWS fornisce una piattaforma infrastrutturale a basso costo, scalabile ed altamente affidabile nel cloud. Uno dei suoi pregi è quello di consentire di avere una grande flessibilità perché ci permette di selezionare il sistema operativo, il linguaggio di programmazione, la piattaforma di applicazione Web e tanti altri servizi. Si ha a disposizione un ambiente virtuale dove si potrà caricare il software e i servizi necessari per la nostra applicazione. I costi sono relativamente bassi e si basano esclusivamente sulla potenza di elaborazione, lo storage e le risorse utilizzate. Si può disporre di una scalabilità molto elevata tramite degli strumenti offerti da AWS, come Auto Scaling e Elastic Load Balancing. Con questi l'applicazione potrà essere ridimensionata e di un ottima affidabilità poiché adotta un approccio alla sicurezza completo che garantisce l'affidabilità dell'infrastruttura sia dal punto di vista fisico che operativo che a livello di software.

EMR è costituito da istanze EC2 che eseguono il lavoro inviato al cluster. Quando si avvia il cluster, Amazon EMR configura le istanze con le applicazioni scelte, ad esempio come nel nostro caso Apache Hadoop o Spark. Inoltre ci permette di scegliere la dimensione e il tipo di istanza più adatti alle esigenze di elaborazione del cluster: elaborazione batch, query a bassa latenza, streaming di dati o archiviazione di grandi quantità di dati, etc. In più, offre una varietà di modi per configurare il software sul cluster, come ad esempio, è possibile installare una versione di Amazon EMR con un set scelto di applicazioni che possono includere strutture versatili, come Hadoop, Hive, Pig, o Spark.

Lo step finale del nostro progetto è stato quello di andare a visualizzare i dati raccolti durante il processo di analisi. Per svolgere questo compito abbiamo utilizzato d3.js. Questa è una libreria scritta in JavaScript utilizzata per creare visualizzazioni dinamiche ed interattive partendo da dati organizzati, visibili attraverso un browser. Per far ciò si serve degli standard Web: SVG, HTML5 e CSS. D3 utilizza funzioni Javascript per selezionare elementi del DOM, creare elementi SVG, aggiungergli uno stile grafico, oppure transizioni, effetti di movimenti e/o tooltip. Questi oggetti possono essere largamente personalizzati tramite i CSS. In questo modo grandi collezioni di dati possono essere facilmente convertiti in oggetti SVG utilizzando semplici funzioni D3 per generare ricche rappresentazioni grafiche di numeri, testi, mappe.

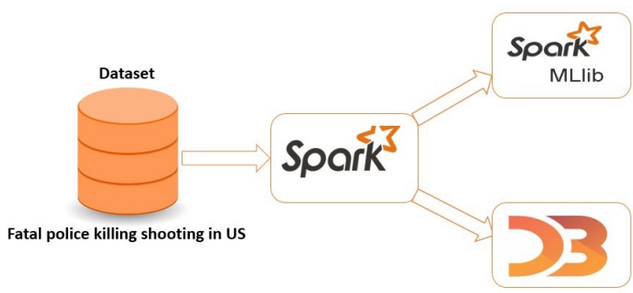


Figure 1: Architettura utilizzata nel progetto di interesse .

# 3 Analisi Effettuate

## 3.1 Analisi del Dataset e Data Cleaning

Prima di iniziare a lavorare con il dataset c'era il bisogno di analizzare come questo fosse strutturato e, come in quasi tutti i dataset, che tipi di problemi potesse contenere. Questo step è di grande importanza ed è necessario in tutti i progetti nell'ambito di data science che lavorano con grandi dataset. In questo modo si valuta il vero numero di ennuple con il quale si può lavorare, e si evitano errori eliminandole o gestendo gli errori a livello di programmazione. Considerando che oltre a fare calcoli statistici si allenerà un modello di classificazione lineare, gli errori presenti nel dataset saranno trasmessi anche al classificatore, danneggiando così la validità dei dati da lui ottenuti. I criteri principali con i quali è stata valutata la qualità del dataset sono:

- Validità.
- Completezza.
- Accuratezza.
- Uniformità (attraverso le diverse tabelle).

I dati contenuti nel dataset sono dati estratti dall'organo di censimento dello stato americano, che essendo un organo ufficiale che si occupa della raccolta di dati numerici di diverso tipo attraverso la popolazione, ci garantisce (anche se non sempre) i criteri sopra elencati. La selezione della provenienza dei dati, cioè la fonte, è di somma importanza visto che nel caso in cui si selezioni una fonte poco affidabile si potrebbe compromettere l'integrità dell'intero progetto.

Oltre alle tabelle descritte sono stati integrati altri dati di supporto per ottenere risultati di rilevanza. In particolare sono stati integrati:

- File CSV con il riferimento al nome dello stato per ogni codice: In ognuna delle cinque tabelle sopra descritte, gli stati sono sempre descritte tramite il codice (per esempio, lo stato della California è descritto come CA). La tabella è stata integrata per visualizzare il nome al posto del codice nei risultati, inoltre in uno dei grafici era necessario il nome.
- File CSV con la stima della popolazione per stato nel 2015: di nuovo, nelle cinque tabelle, non si hanno mai valori sulla popolazione dello stato. Nella tabella *ShareRaceByCity.csv* vengono dati valori percentuali sulla demografica razziale di ogni stato, ma mai i valori totali della popolazione.

Per iniziare a descrivere com'è fatta ogni tabella del dataset partiamo dalla loro cardinalità. La seguente tabella descrive il numero di ennuple che si ha per ognuna delle tabelle:

| **Tabella** | **Nro. di Ennuple** |
|---|---|
| *MedianHouseholdIncome2015* | 29322 |
| *PercentagePeopleBelowPovertyLevel* | 29329 |
| *PercentOver25CompletedHighSchool* | 29329 |
| *PoliceKillingsUS* | 2535 |
| *ShareRaceByCity* | 29268 |

Table 1: Cardinalità delle tabelle del dataset.

Le quattro tabelle che hanno più di 29.000 ennuple hanno tutte quasi la stessa forma. Ogni ennupla è descritta dal seguente schema:

| **Geographic Area (Stato)** | **Città** | **Valore caratteristico della tabella** |
|---|---|---|

Table 2: Schema delle quattro tabelle con più di 29.000 ennuple.

Questo ci permette di unire ognuno di questi valori molto facilmente. Per quanto riguarda la tabella *PoliceKillingsUS.csv* lo schema è completamente diverso. Questa tabella ha quattordici attributi, descritti nella Tabella 3.

Dopo aver analizzato singolarmente ogni tabella sono stati riscontrati i primi problemi. Questi sono descritti per ogni tabella a continuazione:

- *MedianHouseholdIncome2015*: Durante il primo tentativo di parsing delle ennuple di questa tabella, su 29322 sono state parsate con successo 27382 ennuple. Quindi mancavano 1940 ennuple. Analizzando queste ennuple si è scoperto che:
  - 1905 valori non avevano un reddito. Questa è una diretta inconsistenza nei dati, la quale non è risolvibile. Ognuna di queste ennuple è stata filtrata durante il parsing.
  - 24 ennuple non erano state convertite perché mentre il parser si aspettava un dato di tipo Double, si è ritrovato le Stringhe "250.000+", descrivendo il fatto che il reddito era di più di 250.000 dollari, oppure "2.500-" che indicava che il reddito fosse minore ai 2.500 dollari. Tutti questi valori sono stati rispettivamente sostituiti con i valori "250.000" e "2.500".
  - 2 valori contenevano una virgola nel nome, dando problemi al parser che divide i valori ogni volta che incontra una virgola (i file sono tutti CSV), per questo motivo le virgole sono state rimosse per ogni nome.
- *PercentagePeopleBelowPovertyLevel*: Durante il primo tentativo di parsing delle ennuple di questa tabella, su 29329 sono state parsate con successo 29126 ennuple, mancando all'appello 203 ennuple. Analizzando queste ennuple si è scoperto che:

| Attributo | Tipo di Valore | Descrizione |
|---|---|---|
| *Id* | Integer | Chiave primaria dell'ennupla. (Non utilizzata) |
| *Name* | String | Nome della vittima in questione. |
| *Date* | Date | Data dello scontro e morte della vittima. |
| *Manner_of_death* | String | Descrive il modo in cui è morta la vittima. |
| *Armed* | String | Descrive l'arma che aveva la vittima prima del decesso, o se non la aveva. |
| *Age* | Integer | Età della vittima. |
| *Gender* | Character | Sesso della vittima. |
| *Race* | Character | Razza della vittima. |
| *City* | String | Città dov'è successo lo scontro. |
| *State* | String | Codice dello stato dov'è successo lo scontro. |
| *Signs_of_mental_illness* | Boolean | Descrive se la vittima presentava segni di alcun tipo di malattia mentale. |
| *Threat_level* | String | Descrive se la vittima presentava ostilità di alcun tipo. |
| *Flee* | String | Descrive se la vittima stava scappando e il mezzo nel quale scappava. |
| *Body_camera* | Boolean | Descrive se il poliziotto aveva la camera attiva nel momento dello scontro. |

Table 3: Schema della tabella *PoliceKillingsUS.csv* e significato degli attributi.

  - 203 ennuple non avevano il tasso di povertà. Di nuovo, un inconsistenza nei dati non risolvibile, per cui sono state rimosse.
  - 2 valori contenevano una virgola nel nome.

- *PercentOver25CompletedHighSchool*: Durante il primo tentativo di parsing delle ennuple di questa tabella, su 29329 sono state parsate con successo 29130 ennuple, quindi tutte tranne 199 ennuple. Analizzando queste ennuple si è scoperto che:

  - 197 valori non avevano il tasso di scolarizzazione. Inconsistenza nei dati non risolvibile.
  - 2 valori contenevano una virgola nel nome.

- *PoliceKillingsUS*: Durante il primo tentativo di parsing delle ennuple di questa tabella, su 2535 sono state parsate con successo 2310 ennuple, quindi tutte tranne 225 ennuple. Analizzando queste ennuple si è scoperto che:

  - 77 valori non avevano l'età della vittima. Questo rappresenta di nuovo un problema di consistenza nel dataset. Per risolvere problema e non dover eliminare completamente l'ennupla è stata trovata la seguente soluzione: è stato introdotto il valore speciale "-1", in questo modo quando si utilizzano gli altri campi della ennupla non causa problemi, e nel caso di calcoli che comprendono l'età, si filtrano prima tutte le ennuple con valori minori di 1.
  - 147 valori non avevano la razza della vittima. Inoltre, di queste 147 ennuple, 48 non avevano sia l'età che la razza della vittima. Questo problema è stato risolto in modo simile a quello dell'età: Per ognuno di questi valori è stato inserito il carattere "?", e ogni volta che si devono realizzare calcoli relativi alla razza si filtrano le ennuple che lo contengono.
  - 1 ennupla conteneva una virgola nel nome della vittima che è stata eliminata.

- *ShareRaceByCity*: Durante il primo tentativo di parsing delle ennuple di questa tabella, su 29268 sono state parsate con successo 29246 ennuple, quindi tutte tranne 22 ennuple. Analizzando queste ennuple si è scoperto che:

  - 20 ennuple non avevano le percentuali demografiche. Inconsistenza nei dati non risolvibile.
  - 2 valori contenevano una virgola nel nome della città.

Durante un secondo analisi delle tabelle si è riscontrato un nuovo problema, il fatto che nelle quattro tabelle con più di 29.000 ennuple, la maggior parte delle città presentate avevano la parole *"city"* concatenata, mentre nella tabella *PoliceKillingsUS.csv* questa non era presente. Per esempio, la città Adamsville nello stato dell'Alabama, viene descritta come *"Adamsville"* nella tabella appena nominata mentre nelle altre quattro la stessa viene descritta come *"Adamsville City"*. Questo causa problemi quando si realizzano join in base al valore della città tra la prima tabella e le altre quattro. Questo prima problema è stato risolto cercando ogni occorrenza della parola *city* in entrambe le tabelle ed eliminandole.

Dopo aver fatto queste analisi e in particolare dopo aver trovato quest'ultimo problema, si è resa chiara una cosa molto importante da tenere in conto, cioè il fatto che le quatto tabelle sempre nominate insieme hanno la stessa fonte di provenienza, mentre la tabella *PoliceKillingsUS.csv* ha dei dati che sono stati probabilmente raccolti da altri organi. Ciò si capisce dal fatto che in ognuna delle quattro tabelle si avevano le stesse due ennuple con la virgola nel nome. Inoltre l'inconsistenza semantica che si ha nel nome della città lo rende chiaro.

## 3.2 Task di base

In questa sotto-sezione vengono descritte le analisi svolte sui vari csv, descritti poc'anzi, implementate all'interno del progetto.

### 3.2.1 Poorest States

Il primo task di interesse è stato quello di andare ad analizzare la distribuzione del tasso di povertà nei vari stati, di seguito segue lo pseudocodice per la creazione dell'RDD.

```
method calculatePOorestStates(rdd)

    state2cityPoverty <- PairRDD del tipo <String, Double> ottenuto
                         dall'applicazione di un map To pair ad rdd
                         dove si pone come chiave lo stato e come
                         valore il tasso di povert  del suddetto
                         stato ed una reduce by key per fare la somma
                         del tasso di povert  per la citt  appartenente
                         a quello stato.

        state2CityNum <- PairRDD del tipo <String, Integer> ottenuto
                         dall'applicazione di un map To pair ad rdd
                         dove si mette come chiave lo stato e come
                         valore un count 1 e si applica una reduce by
                         key per sommare tutti gli uno, ci  serve per
                         contare il numero di citt  per stato.

        state2meanPov <- PairRDD ottenuto dall'applicazione di un join
                        per prendere i dati calcolati negli RDD
                        precedenti ed un map To pair dove si mette come
                        chiave il nome dello stato e come valore il
                        calcolo della media tra il tasso di povert  per
                        ogni citt  e il numero di citt  prese
                        in considerazione.

        sorted <- RDD del tipo <Tuple2<String, Double>> ottenuto
                  dall'applicazione di una map sull'RDD state2meanPov
                  dove si mette come chiave il nome dello stato e come
                  valore il valore della media calcolata poc'anzi, si
                  applica poi una sortBy dove si ordina per valore in
                  ordine decrescente quindi in output si visualizzer
                  dallo stato pi  povero a quello meno povero.

        return sorted
```

### 3.2.2 Most Common Victim Name

Il task che segue si prepone come obiettivo quello di andare a ricercare tra le tante vittime dei nomi che spiccano rispetto ad altri per ricorrenza. Quello che segue è l'RDD per lo svolgimento della suddetta analisi.

```
method mostCommonNameVictim(rdd)

        name2count <- PairRDD del tipo <String, Integer> ottenuto
                      dall'applicazione di un map To pair ad rdd dove
                      si pone come chiave il nome e come un count a 1
                      ed una reduce by key per fare la somma di tutte
                      le volte che quel nome compare.

        sorted <- RDD del tipo <Tuple2<String,Integer>> ottenuto
                  dall'applicazione di un map a name2count ed un
                  sortBy dove si ordina per valore in maniera
                  decrescente, quindi in output si avr  una lista
                  ordinata dal nome pi  comune a quello meno comune.

        return sorted
```

### 3.2.3 State mean High School graduates

Il task che segue si prepone come obiettivo quello di andare ad individuare per ogni stato la media delle persone con più di 25 anni che hanno un diploma. Quello che segue è l'RDD per lo svolgimento della suddetta analisi.

```
method calculateStateMeanEducation(rddHS)

        state2cityPerc <- PairRDD del tipo <String, Double> ottenuto
                          dall'applicazione di un map To pair ad rdd
                          dove si pone come chiave lo stato e come valore
                          la  percentuale di diplomati per stato.
```

```
        state2count <- PairRDD del tipo <String, Integer> ottenuto dall'
                       applicazione di un map To pair ad rdd dove si
                       mette come chiave lo stato e come valore un
                       count 1 e si applica una reduce by key per sommare
                       tutti gli uno, ci  serve per contare il numero di
                       persone che hanno completato gli studi per stato.

        state2meanHS <- PairRDD otteuto dall'applicazione di un join per
                        prendere i dati calcolati negli RDD precedenti
                        ed un map To pair dove si mette come chiave il nome
                        dello stato e come valore il calcolo della media
                        della percentuale di diplomati.

        return state2meanHS
```

#### 3.2.4 Killed people by state

Con questa analisi invece, si è preposti come obiettivo quello di andare ad individuare gli stati più colpiti da atti criminosi, nel dettaglio in questo caso per atti criminosi intendiamo le uccisioni che avvengono per mano della polizia, andando a restituire in output una lista ponendo come primo elemento lo stato che ha subito più scontri letali. Quello che segue è lo pseudocodice che da un'idea su cosa è stato fatto per svolgere la suddetta analisi.

```
method calculateKilledPeopleByState (rdd)

        state2count <- PairRDD del tipo <String, Integer> ottenuto
                       dall'applicazione di un map To pair ad rdd
                       dove si pone come chiave lo stato e come
                       valore un count a 1 ed una reduce by key
                       per fare la somma di tutte le volte che quello
                       stato compare.

        sorted <- RDD del tipo <Tuple2<String,Integer>> ottenuto
                  dall'applicazione di un map a name2count ed
                  un sortBy dove si ordina per valore in maniera
                  decrescente, quindi in output si avr  una lista
                  ordinata dallo stato con pi  uccisioni avvenute per mano della polizia a q

        return sorted
```

#### 3.2.5 Education VS Poverty

Con la prefissione di questo task siamo andati ad effettuare un confronto tra il tasso di scolarizzazione e il tasso di povertà per ogni stato. Grazie ai risultati ottenuti si è notato come i due parametri sono inversamente proporzionali, ovvero più il tasso di scolarizzazione è alto più il tasso di povertà è basso. Quello che segue è la costruzione dell'RDD per risolvere il task di interesse, come si noterà sono stati utilizzati degli RDD costruiti nei task precedenti.

```
method calculateEducationVSPoverty( rddHS, rddPP)

        state2meanHS <- calculateStateMeanEducation(rddHS) crea
                        un PairRDD del tipo <String,Double>
tramite un metodo implementato in precedenza.

        state2poorness <- calculatePoorestStates(rddPP) crea un PairRDD
                          tramite anche l'applicazione di un map to pair
                          applicato a calculatePOosrestStates per mappare
                          le varie tuple.

        state2educ2poor <- PairRD del tipo <String, Tuple2<Double, Double>
                           ottenuto dall'applicazione di un join che va
                           a prendere in considerazione i dati degli
                           RDD calcolati in precedenza.

        return state2educ2poor
```

#### 3.2.6 Killed people by race

Con questo task l'obiettivo che ci siamo preposti è quello di andare a calcolare quante persone sono state uccise durante degli scontri con la polizia, ed andare a classificare e suddividere queste persone per razza. Quello che segue è lo pseudocodice per la risoluzione del suddetto task.

```
method calculateKilledPeopleByrace(rdd)

        race2count <- PairRDD del tipo <Character, Integer> ottenuto
                      dall'applicazione di un map To pair ad rdd
```

```
                dove si pone come chiave la razza e come valore
                un count a 1 ed una reduce by key per fare la
                somma di tutte le volte che quella razza compare.

    sorted <- RDD del tipo <Tuple2<Character,Integer>> ottenuto
              dall'applicazione di un map a race2count ed un
              sortBy dove si ordina per valore in maniera
              decrescente, quindi in output si avr  una lista
              ordinata della razza pi  colpita a quella meno colpita.

    return sorted
```

#### 3.2.7 Calculate victims to population proportion

Con questa analisi l'obiettivo è quello di andare a calcolare il numero di vittime per ogni stato in proporzione alla popolazione dello stato di interesse. Di seguito l'RDD per la risoluzione del nostro job.

```
method calculateVictimsToPopulationProportion( rddPK, rddSP)


    state2victimCount <- calculateKilledPeopleByState(rddPK)
                         creo un PairRDD del tipo <String,
                         Integer> ottenuto dall applicazione
                         di un map to pair a calculateKilledPeopleByState

    state2Population <- PairRDD del tipo <String,Integer>
                        ottenuto dall'applicazione di un map To
                        pair a rddSP dove come chiave mettimao
                        lo stato e come valore la popolazione.

    state2prop <- PairRDD del tipo <String,Double> ottenuto dal
                  join di due PairRDD creati in precedenza
                  state2victimCount e state2Population e dall'
                  applicaizone di un map to pair dove si mette
                  come chiave il nome dello stato  e come valore
                  la proporzione delle vittime in base alla
                  popolazione totale.

    sorted <- RDD del tipo <Tuple2<String, Double>> ottenuto
              dall'applicazione di un map su state2prop in cui
              si ha come chiave lo stato e come valore la proporzione
              di vittime in base alla popolazione totale e si
              fa un sortBy in ordine decrescente al valore in modo
              tale da visualizzare i risultati dagli stati con pi
              vittime in base alla popolazione a quello con meno
              vittime sempre in basea alla popolazione.

    return sorted
```

#### 3.2.8 Common manner of death

Con lo svolgimento di questo task il Goal che ci siamo preposti è quello di analizzare nel dettaglio il come avvengono queste morti per mano della polizia. Come si vedrà dai risultati le modalità saranno principalmente due. Di seguito la costruzione dell'RDD per la risoluzione del nostro Job.

```
method calculateMostCommonMannerOfDeath(rdd)

    typeOdDeath2count <- PairRDD del tipo <String,Integer> ottenuto
                         dall'applicazione di un map To Pair su rdd dove
                         come chiave si mette come si     morti e come valore
                         un count a 1 per contare quante volte un tipo
                         di morte si ripete e si fa una reduce by key per
                         sommare il count.

    sorted <- RDD del tipo <Tuple2<String,Integer>> ottenuto
              dall'applicaizone di un map su typeOfDeath2count
              ed un sort By sul valore, cosi da ottenere in output
              una lista in ordine decrescente dalla morte pi
              comune a quella meno comune avvenuta per mano della
              polizia.

    return sorted
```

### 3.2.9 Dangerous Cities

Come si può intuire questa sotto-sezione si concentrerà sempre sull'analisi delle morti avvenute per mano della polizia però concentrandoci questa volta sulle città in cui avviene l'evento, tutto ciò per capire se vi è una città in cui queste disgrazie si presentano più spesso rispetto ad altre.

```
method dangerousCities(rdd)

        nameOfCities <- PairRDD del tipo <String,Integer> ottenuto
                        dall'applicazione di un map To Pair su rdd
                        dove come chiave si mette la citt  e come valore
                        un count a 1 per contare quante volte compare
                        la citt  e si fa una reduce by key per sommare
                        il count.

        sorteed <- RDD del tipo <Tuple2<String,Integer>> ottenuto
                   dall'applicaizone di un map su nameOfCities ed un
                   sort By sul valore, cosi da ottenre in output una
                   lista in ordine decrescente dalla  citt  pi  pericolosa
                   a quella meno.

        return sorted
```

### 3.2.10 Gender of victim

Con questo task vogliamo vedere se vi è tra le vittime un sesso che è più colpito rispetto all'altro. Quello che segue è l'RDD per la risoluzione del job.

```
method genderOfVictim(rdd)

        gender <- PairRDD del tipo <Character,Integer> ottenuto
                  dall'applicazione di un map To Pair su rdd
                  dove come chiave si mette il genere della vittima
                  e come valore un count a 1 per contare quante volte
                  si ha quel genere (maschio  o femmina) e si fa un
                  reduce by key per sommare il count.

        sorteed <- RDD del tipo <Tuple2<Character,Integer>> ottenuto
                   dall'applicaizone di un map su gender ed un sort By
                   sul valore, cosi da ottenre in output una lista in
                   ordine decrescente dal genere pi  colpito a qeullo meno.

        return sorted
```

### 3.2.11 Mean age by race

Questo job è stato preposto per andare ad analizzare, sempre nell'ambito delle uccisioni per mano della polizia, l'età media di ogni razza per capire se vi è una fascia di età maggiormente colpita rispetto alle altre. Questo ragionamento è stato ampliato ad ogni razza passando dai Nativi americani agli Asiatici. Quello che segue è l'RDD per la risoluzione del task preposto.

```
method calculateMeanAgeWithRace(rdd)


        age4race <- PairRDD del tipo <Character,Integer> ottenuto
                    dall'applicazione di un filter ad rdd per
                    rimuovere le et  non valide < di 1, si
                    effettua l'applicazione di un map to pair
                    dove si pone come chiave la razza e come valore
                    un count per contare quante volte compare una
                    determinata razza, ed infine si applica una reduce
                        by key per sommare il count posto come valore
                        in precedenza.

        numVictim4race <- PairRDD del tipo <Character,Integer>
                          che si ottiene dall'applicazione
                                  di un filter applicato ad rdd, come
                                  prima per filtrare le et  da non prendere
                                  in considerazione e dall'applicazione di
                                  un map to pair dove si pone come chiave
                                  la razza e come valore l'et  e di fa una
                                  reduce by key cos  da avere l'et  complessiva
                                  per quella razza.
```

```
meanAge4race <- PairRDD del tipo <Character,Double> ottenuto
                dall'applicazione di un join per
                        usisfruire dei dati raccolti dai due RDD
                        precedenti e dall'applicaizone di un
                        map to pair dove si pone come chiave la
                        razza e come valore il calcolo della della
                        media dell'et  per una determinata razza.

sorted <- RDD del tipo <Tuple2<Character, Double>> ottenuto
          dall'applicazione di una funzione di map a
          menaAge4race e di un sort By dove si ordina
          in maniera decrescente in base al valore, quindi
          in output si avr  una lista di razza ed et  media
          ordinata in maniera decrescente in base alla media dell'et .

return sorted
```

#### 3.2.12 Most Common Weapon Used

In questa sezione il task di interesse si pone come obiettivo quello di andare a ricercare l'arma più ricorrente che le vittime avevano nel momento in cui sono state uccise da un membro del corpo poliziesco. Quello che segue è l'RDD per la risoluzione del Job.

```
method commonWeaponUsed(rdd)

commonWeapon <- PairRDD del tipo <String,Integer> ottenuto
                dall'applicazione di un map to pair a rdd
                dove si effettua un count e si pone come chiave
                l'arma e come valore un 1 per identificare
                quante volte compare quell'arma e una reduce
                        by key per sommare il numero di volte complessive
                        che compare quell'arma.

sorted <- RDD del tipo <Tuple2<String,Integer>> ottenuto
          dall'applicazione di un map a commonWeapon e
          un sort By per ordinare in maniera decrescente
          in baser al valore quindi in output si avr
          una lista ordinata dall'arma pi  presente
          a quella meno presente.

return sorted
```

#### 3.2.13 Body camera Check

In questo task si prende in considerazione il campo *body_camera* nella tabella *Police_KillingsUS.csv*, campo che ci informa se il poliziotto che ha ucciso il soggetto era munito o meno di body camera, fondamentale per andare ad analizzare e studiare la dinamica dell'evento. Qui l'RDD per la risoluzione del Job.

```
method bodyCameraCheck(rdd)

countBodyCamera <- PairRDD del tipo <Boolean,Integer> ottenuto
                   dall'applicazione di un map to pair a rdd
                   dove si effettua un count e si pone come chiave
                   se si ha la body camera o meno e come valore
                   un 1 per identificare quante volte la situazione
                   di ripete e una reduce by key per sommare
                   il numero di volte complessive che si ha o meno la bodycamera.

sorted <- RDD del tipo <Tuple2<Boolean,Integer>> ottenuto
          dall'applicazione di un map a countBodyCamera e
          un sort By per ordinare in maniera decrescente in
          base al valore quindi in output si avr  una lista
          ordinata per capire se sono state pi  le volte che
          il poliziotto aveva la body camera o meno.

return sorted
```

#### 3.2.14 Share race by state

Questo task è molto importante poiché ci permette di andare ad analizzare la distribuzione di ogni razza in ogni stato, questo lavoro è stato svolto andando a calcolare la media di ogni razza per ogni città per fare poi una media complessiva per ogni stato così da avere un'idea sulla distribuzione di ogni razza all'interno di ogni stato. Per svolgere questo lavoro sono stati costruiti cinque RDD differenti per le cinque razze e un RDD finale per mettere insieme il tutto, per sintesi verrà mostrato lo pseudocodice di una singola razza nel dettaglio quello della razza Asiatica e lo pseudocodice finale.

```
method shareRaceAsianByState(rddSR)

        countAsian <- PairRDD del tipo <String,Integer> ottenuto
                      dall'applicazione di una map to pair
                          a rddSR dove si mette come chiave lo stato
                          e come valore un count per capire quante
                          volte lo stato compare in quel dataset e una
                          reduce by key per fare la somma di tutti gli 1.

        stateAsian <- PairRDD del tipo <String,Double> ottenuto
                      dall'applicazione di una mapToPair dove si
                      mette come chiave lo stato e come valore la
                      media della presenza della razza asiatica
                      per quello stato

        meanRaceAsian4state <- PairRDD del tipo <String,Double> ottenuto
                               tramite un join tra i due PairRDD creati
                               in precendeza dove si applica una mapToPair
                               in cui si mette come chiave lo stato e
                               come valore la media della distribuione
                               della razza asiatica per quello stato

        return meanRaceAsian4state
```

Quello che segue invece è l'RDD per poter mettere insieme il tutto.

```
method unionRace4state(rddSR)

        asian4state <- PairRDD del tipo <String,Double> ottenuto
                       da un metodo denominato shareRaceAsianByState(rddSR)

        white4state <- PairRDD del tipo <String, Double> ottenuto da
                       un metodo denominato shareRaceWhiteByState(rddSR)

        black4state <- PairRDD del tipo <String, Double> ottenuto da
                       un metodo denominato shareRaceBlackByState(rddSR)

        nativeAmerican4state <- PairRDD del tipo <String, Double> ottenuto da
                                un metodo denominato
                                shareRaceNativeAmericanByState(rddSR)

        hispanic4state <- PairRDD del tipo <String, Double> ottenuto da
                          un metodo denominato
                          shareRaceHispanicByState(rddSR)


        joinAsiaWhite <- PairRDD del tipo <String, Tuple2<Double,Double>
                         che si ottiene dall'applicazione di un join
                         tra asin4stae che white4state

        joinAsWhBlack <- PairRDD del tipo <String, Tuple2<Double
                        ,DOuble>,Double> che si ottiene dal join tra
                        joinAsianWhite e black4state


        joinAsWhBlNativeAmerican <- PairRDD del tipo <String,
                                    Tuple2<Tuple2<Tuple2<Double,Double>
                                    ,DOuble>,Double>> ottenuto dall'
                                    applicazione di un join tra joinAsWhBlack
                                    e nativeAmerica4state.

        joinAll <- RDD del tipo <String,Tuple2<Tuple2<Tuple2
                   <Tuple2<Double,Double>, Double>,Double>,
                           Double>> ottenuto dal join tra joinAsWhBlNativeAmerican
                           e hispanic4state

        return joinAll
```

#### 3.2.15 Killed by month

Con quest'ultimo task ci siamo preposti di andare ad analizzare sempre le uccisioni però questa volta nell'arco temporale dell'anno per andare a vedere se la distribuzione delle uccisioni è uniformi nei vari mesi o no. Qui è presente lo pseudocodice per la risoluzione di quest'ultimo task.

```
method killedByMonth(rdd)

        monthWithMostKill <- PairRDD del tipo <Integer,Integer> ottenuto
                             dall'applicazione di un map to pair a rdd
                             nel quali si mette come chiave il mese e come
                             valore un  count che       con un reduce by key
                             conta quante volte un determinato mese
                             compare nel dataset

        sorted <- RDD del tipo <Tuple2<Integer,Integer>> ottenuto
                  dall'applicaizone di un map a montWithMostKill
                  e un sortBy per ottenere in output una lista ordinata
                  in maniera decrescente dal mese con pi  kill a quello
                  con meno, fatto per monitorare l'attivit  criminosta
                  durante l'anno.

        return sorted
```

## 3.3 Modello di Classificazione di Machine Learning

In questa sezione sarà mostrato com'è stato realizzato il modello di classificazione tramite Apache MLlib.

Uno degli scopi di questo progetto era quello di testare le capacità di un modello di classificazione appoggiandosi all'analisi del dataset realizzato con Spark. L'obiettivo perseguito era quello di realizzare un classificatore che potesse predire la razza della vittima in base ad altre caratteristiche. A questo punto il lavoro si suddivideva in due parti principali, la selezione delle feature da utilizzare per il modello, la costruzione del classificatore.

### 3.3.1 Feature Selection

Durante la progettazione di qualsiasi tipo di modello di machine learning uno degli step più importanti e critici è quello della selezione delle feature. Se svolto male, questo step compromette completamente l'efficacia del modello, indipendentemente dal lavoro svolto negli step successivi.

Le due idee principali dietro la selezione delle feature sono:

- La rimozione di feature che possono essere considerate irrilevanti o ridondanti ai fini degli obiettivi del modello.
- La selezione di feature caratteristiche che siano di rilevanza per la classe che si vuole predire.

Per la selezione delle feature solitamente si applicano algoritmi che permettono di realizzare prove per capire quali feature apportano più risultati al modello. Metodi come All Subset o Forward Stepsize sono algoritmi iterativi che analizzano le possibili combinazioni di feature verificando le prestazioni ottenute con ognuna di esse. Questo approccio è quello ideale quanto nel modello si hanno tante feature e si hanno a disposizione librerie per applicarlo. Nel caso di questo progetto però la selezione non è stata fatta in questo modo. Ciò è stato motivato dalle seguenti precisazioni:

1. Essendo un progetto più incentrato sull'utilizzo di tecnologie come Spark, si è preferito fare l'analisi delle feature "a mano" utilizzando questo framework.
2. La libreria utilizzata per gli algoritmi di machine learning, cioè MLlib, non conteneva nessun tipo di algoritmo per la selezione delle feature.
3. L'algoritmo poteva essere integrato manualmente, si è deciso però di non farlo dato che il numero di feature da considerare non era così elevato e visto che era già stato svolto il lavoro di analisi con Spark, si aveva già un idea di quali feature potessero essere di rilevanza.

A questo punto consideriamo la seguente tabella che indica il numero di feature da considerare per il modello.

| **Tabella** | **# Feature** |
|---|---|
| *MedianHouseHoldIncome2015* | 3 |
| *PercentOver25CompletedHighSchool* | 3 |
| *PercentagePeopleBelowPovertyLevel* | 3 |
| *PoliceKillingUS* | 13 |
| *ShareRaceByCity* | 7 |
| | Totale: 29 |

Table 4: Tabella iniziale delle feature.

Ventinove feature possono sembrare tante, però la maggior parte di queste possono essere escluse, partendo dalle due feature più ricorrenti: *State* e *City*. Il nome della città e dello stato dove si è svolto lo scontro non ha assolutamente nessuna importanza. Ciò che conta veramente sono le caratteristiche intrinseche di questi due, cioè le feature relative al posto. Per questo motivo queste, anche se saranno usate per fare il join tra le tabelle per unire le informazioni in un unica ennupla, non saranno integrate nel modello.

Adesso parliamo esclusivamente delle colonne presenti nella tabella *PoliceKillingUS*. Di queste tredici feature, tre condividono un problema che non permette che queste siano utilizzate nel modello. Detto meglio, possono essere utilizzate ma ne deteriorerebbero la qualità. Queste tre colonne sono:

- *Gender*.

- *Manner_of_death*.
- *Body_camera*.

Il problema che condividono è la scarsa distribuzione di valori. Una feature, per essere di interesse deve avere una distribuzione di valori caratteristica tra i possibili valori univoci. Questo non è il caso con queste tre feature. Andando nel dettaglio troviamo che per quanto riguarda il sesso, cioè *Gender*, 96% delle vittime che si hanno sono di sesso maschile mentre il restante 4% è di sesso femminile. Per *Manner_of_death*, il caso è simile, 93% delle ennuple hanno come valore "shot", cioè sono morte dalla sparatoria, mentre il restante 7% ha come valore "shot and Tasered". L'ultima colonna, contenente valori booleani, ha 89% dei valori "falsi" e il restante 11% veri, indicando il fatto che pochi poliziotti avevano la camera attiva durante lo scontro, rendendola una feature irrilevante.

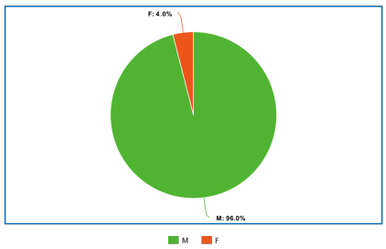


Figure 2: Gender.


Figure 3: Manner_of_death.

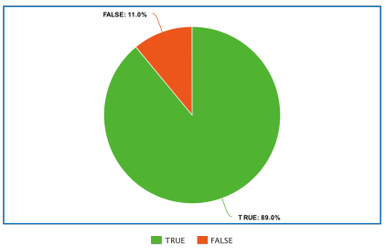


Figure 4: Body_camera

Esistono altre due colonne dalle quali non si possono estrarre informazioni di interesse, rendendole obsolete per il modello. Queste sono *Id* e *Date*. Per quanto riguarda la prima, essendo questo un numero auto-generato e univoco per ogni ennupla, non ha nessuna importanza ne pattern di interesse. Per quanto riguarda invece la seconda, si ha un caso simile a *City* e *State*, cioè, il fatto che in se la data non è di importanza ma i dati relativi alle date lo sono. Per fare un esempio, il fatto che una vittima è stata uccisa il 27 luglio del 2015 non ha nessuna importanza, a meno che si rilevi che nel nel luglio del 2015 per qualche motivo si sono incrementate in maniera rilevante le morti di persone asiatiche. Dopo aver fatto questo tipo di analisi e non ritrovare nessun pattern, la colonna è stata esclusa.

A questo punto si è fatto uno studio su altre tre colonne: *Flee*, *Threat_level* e *Signs_of_mental_illness* ma dopo averle analizzato e dopo aver fatto diverse prove con il classificatore si è concluso che queste non miglioravano la predizione e in alcuni casi la peggioravano.

Infine, la la tabella vista prima a questo punto è diventata

| **Tabella** | **# Feature** |
|---|---|
| *MedianHouseHoldIncome2015* | 1 |
| *PercentOver25CompletedHighSchool* | 1 |
| *PercentagePeopleBelowPovertyLevel* | 1 |
| *PoliceKillingUS* | 3 |
| *ShareRaceByCity* | 5 |
| | Totale: 11 |

Table 5: Tabella finale delle feature.

Inoltre, le singole ennuple utilizzate hanno i seguenti campi:

1. Age
2. Share_White
3. Share_Black
4. Share_Hispanic
5. Share_Asian
6. Share_Native
7. Percent_HighSchool
8. Percent_Poverty
9. Median_Income
10. Armed
11. Name (solo il primo nome è stato filtrato)

Si noti però che questa non è la versione finale delle feature scelte, semplicemente la versione finale dopo il primo step.

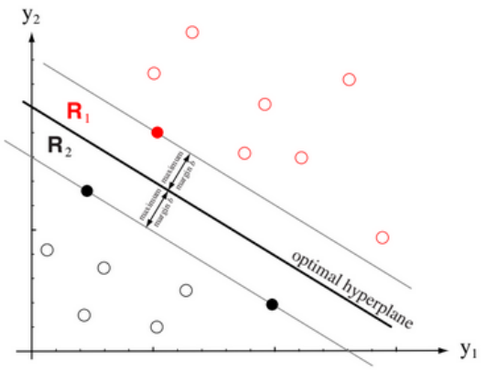


Figure 5: Esempio grafico di un SVM.

### 3.3.2 Classificatore Multiclasse

A questo punto è arrivato il momento di progettare le caratteristiche del classificatore. Per svolgere la predizione è stato deciso di utilizzare un classificatore multiclasse, anche detto SVM Multiclasse.

Le Support Vector Machine sono un modello di apprendimento supervisionato, che dato un insieme di ennuple o dati etichettati, quindi aventi una classe di appartenenza, si occupa di costruire un modello che assegni una delle due classi ai nuovi esempi, ottenendo quindi un classificatore lineare. Di base, con due classi di pattern multidimensionali linearmente separabili, tra tutti i possibili iperpiani di separazione, un SVM determina quello in grado di separare le classi con il maggior margine possibile. Durante l'allenamento, per ogni ennupla che gli viene passata, si occupa di aggiustare la posizione e l'inclinazione del margine di separazione, fino ad esaurire il dataset.

Questa è la soluzione adottata nel caso di un classificatore binario. Nel nostro caso abbiamo bisogno di un modello che sia in grado di predire la razza di una vittima tra cinque razze, cioè che si in grado di gestire problemi con più di due classi. La soluzione adottata è stata quella del classificatore multiclasse OneVsAll o OneVsRest. Questo classificatore opera nel seguente modo:

Date s classi $w_1$,..., $w_s$, per ogni classe $w_k$ si determina con una SVM lineare la superficie di separazione tra i pattern di $w_k$ da una parte e i pattern di tutte le classi rimanenti $w_h$ con $h \neq k$ dall'altra parte. In questo modo si ottiene la funzione $D_k(x)$ che indica quanto x è distante dalla superficie decisionale in direzione di $w_k$. Maggiore è $D_k(x)$ e più confidenti siamo che il pattern $x$ appartenga alla classe $w_k$. A questo punto alla fine dell'addestramento si assegna $x$ alla classe $k^*$ per cui è massima la distanza della superficie decisionale:

$$k^* = \arg\max_k \{D_k(x)\}$$

con $x$ pattern da classificare e $k = 1, 2, ...s$. Si noti che quindi vengono eseguiti $s$ training.

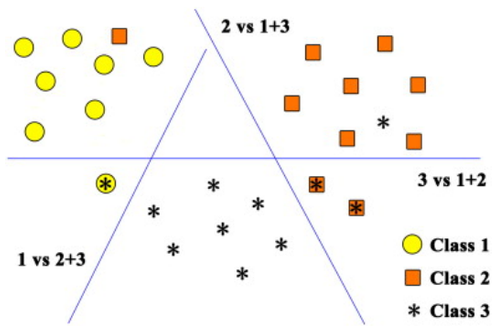


Figure 6: Esempio grafico di un SVM Multiclasse One vs Rest.

Prima di andare avanti dobbiamo specificare un ulteriore operazione realizzata con le feature. Visto che un modello binario non sa interpretare stringhe ossia dati categorici, dobbiamo gestire due feature che appartengono al nostro modello per trasformarle in un formato che il classificatore "capisca", cioè numerico. Le due feature di cui parliamo sono *Armed* e *Name*. Per questo motivo utilizziamo la One Hot Encoding, che trasforma tutti i dati categorici in dati numerici.

| Color |
|---|
| Red |
| Red |
| Yellow |
| Green |
| Yellow |

| Red | Yellow | Green |
|---|---|---|
| 1 | 0 | 0 |
| 1 | 0 | 0 |
| 0 | 1 | 0 |
| 0 | 0 | 1 |

Figure 7: Trasformazione dei valori di una feature con One Hot Encoding.

Inoltre le label delle classi, cioè la razza delle vittime, sono state convertiti da caratteri a numeri. Sempre per il motivo sopra elencato.

A questo punto è tutto pronto per eseguire il modello. Il file di input che questo riceve ha un formato standard, quello dettato da LIBSVM, libreria molto utilizzata nell'ambito delle Support Vector Machines. Ogni ennupla, in questo formato è quindi scritta nel seguente modo:

Label 1:Valore_Feature1 2:Valore_Feature2 ... N:Valore_FeatureN

Dopo aver realizzato tutti i join filtrando ennuple con valori mancanti ed erronei nel file sono state scritte 1660 ennuple, esse costituiscono il nostro dataset per il modello. Questo purtroppo è negativo visto che per allenare un buon modello di classificazione servono moltissimi più dati. Ciò andrà a influire in modo critico nella precisione del sistema, ma purtroppo questo è un limite imposto dal dataset che si aveva a disposizione (se al posto di 1600 record se ne avevano 2500 come nel dataset iniziale la situazione non sarebbe cambiata). Inoltre dopo aver applicato la One Hot Encoding ogni ennupla ha esattamente 715 features. Questo numero è dato dalle prime nove feature prima elencate, inoltre, le seguenti sessantanove sono tutti i possibili valori della colonna *Armed* del dataset, e tutte le restanti sono tutti possibili valori della colonna *Name*, cioè i possibili primi nomi.

Il nostro modello quindi splitterà il dataset di 1660 ennuple in due:

- 70% sarà utilizzato come set di addestramento o training set.
- 30% sarà invece utilizzato come test set.

Ad ogni esecuzione lo split realizzato prende ennuple casuali, motivo per il quale i risultati della precisione variano ad ogni run del modello. In più c'è da dire che il modello si addestra per dieci epoche prima di iniziare a predire.

### 3.3.3 Risultati

Prima di ottenere il risultato finale, diversi modelli sono stati realizzati con diverse combinazioni delle feature sopra descritte. Alla fine discuteremo solo di due modelli che sono stati individuati come rilevanti. Il primo perché contiene le feature "essenziali", il secondo perché è quello che ha raggiunto la precisione più alta.

Il primo modello ha raggiunto i seguenti risultati:

$$Precision\ Score = 67\%$$

$$Error\ Rate = 33\%$$

La cosa interessante di questo modello è che in confronto al prossimo, dove la precisione raggiunta non è molto più elevata, questo contiene solo le prime sei feature prima elencate, cioè:

- Age
- Share_White
- Share_Black
- Share_Hispanic
- Share_Asian
- Share_Native

Queste sei feature (che in realtà corrispondono a due feature, l'età della vittima e la demografia razziale della città) sono state determinate come le feature essenziali del modello. In particolare, per quanto riguarda *Age*, questo si era inteso dall'analisi realizzata per la selezione delle feature fatta in precedenza, data la distribuzione caratteristica che si ha per ogni razza, rappresentata nel seguente grafico.

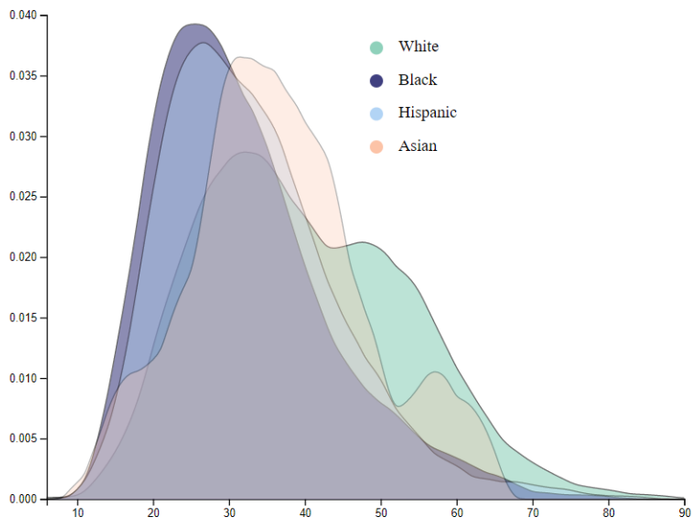


Figure 8: Distribuzione dell'età delle vittime per ogni razza tranne Native American.

Per quanto riguarda il secondo modello, cioè quello che ha ottenuto la precisione maggiore ha ottenuto i seguenti risultati:

$$Precision\ Score = 69.3\%$$

$$Error\ Rate = 30.7\%$$

Questo risultato è stato ottenuto utilizzando le seguenti feature:

- Age
- Share_White
- Share_Black
- Share_Hispanic
- Share_Asian
- Share_Native
- Percent_HighSchool
- Percent_Poverty
- Armed (One Hot)
- Name (One Hot)

Utilizzando quelle quattro feature aggiuntive si è riusciti a raggiungere una precisione con solo il 2% in più rispetto al primo modello. Per questo motivo quelle due feature sono state ritenute essenziali per il miglioramento del modello.

Infine, i risultati non sono stati molto elevati, ma non ci si aspettava di più data la piccolezza del dataset a disposizione.

# 4 Risultati e Visualizzazione

Al termine delle varie analisi effettuate si è arrivati a delle conclusioni con una certa rilevanza. Si è notato che:

- I neri muoiono 3 volte in più rispetto ai bianchi per scontri con la polizia.
- Il tasso di povertà degli stati è inversamente proporzionale al tasso di scolarizzazione.
- In media i morti delle etnie nere e ispaniche (31 e 33) tendono a morire ad un età più giovane rispetto ai bianchi (40).
- Il modo più comune di essere armati è tramite un arma da fuoco (più della metà delle vittime ne possedeva una al momento) (55%).
- In 89% dei casi il poliziotto coinvolto nello scontro aveva la body cam spenta quando per legge deve essere sempre accesa.
- 1 su ogni 4 vittime presentava segni di disturbi mentali.
- Lo stato più pericoloso è il New Mexico che a sua volta è compreso nei 9 stati con tasso di scolarizzazione più basso (81%), ed è il 4to stato con tasso di povertà più elevato (23%).
- 3.86% delle vittime (98) sono morte mentre erano disarmate e non scappavano, e di queste il 32% delle vittime erano nere, 21% ispaniche e 41% bianche. Solo in 16% di questi casi, il poliziotto aveva la body cam accesa per confermare i fatti.
- Sui 104 casi di vittime di sesso femminile (4.22%), il 55% erano bianche e il 25% nere. Il 46% erano armate con arme da fuoco rispetto al 55% nelle vittime maschili.

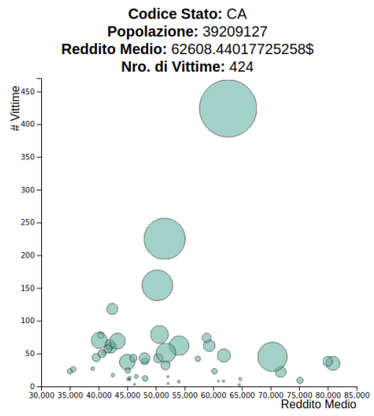


Figure 9: Morti in proporzione alla popolazione dello stato CA e al reddito medio.

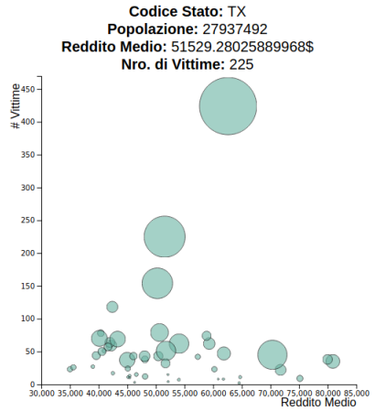


Figure 10: Morti in proporzione alla popolazione dello stato TX e al reddito medio.

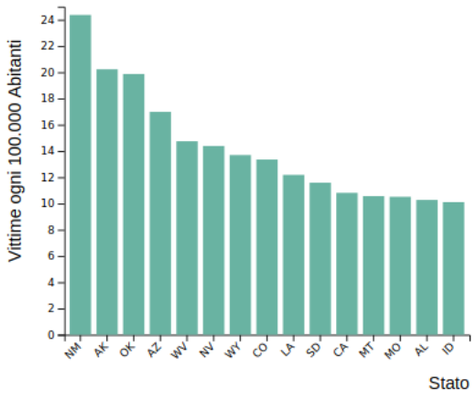


Figure 11: Stati con maggior tasso di criminalità.


Figure 12: Distribuzione dell'età per genere.

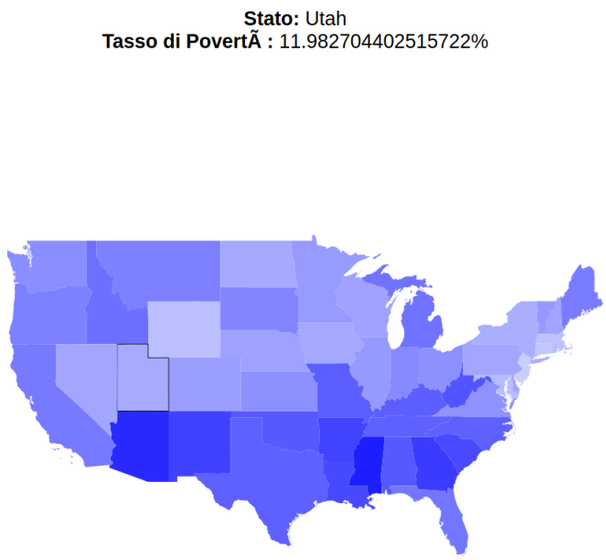


Figure 13: Visualizzazione del tasso di povertà per stato (Utah).

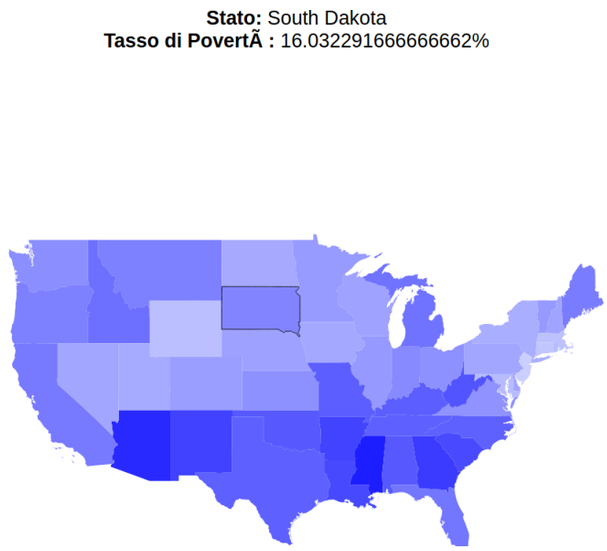


Figure 14: Visualizzazione del tasso di povertà per stato (South Dakota).

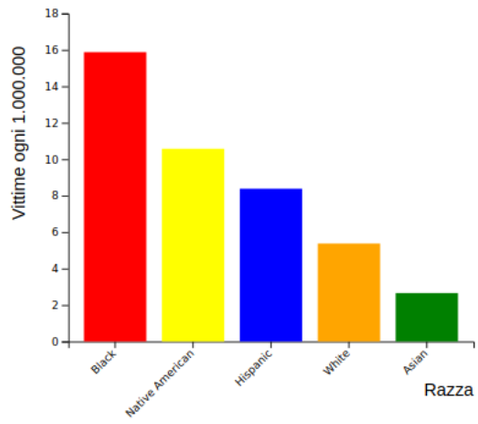


Figure 15: Numero di vittime in proporzione al numero di integranti della razza.

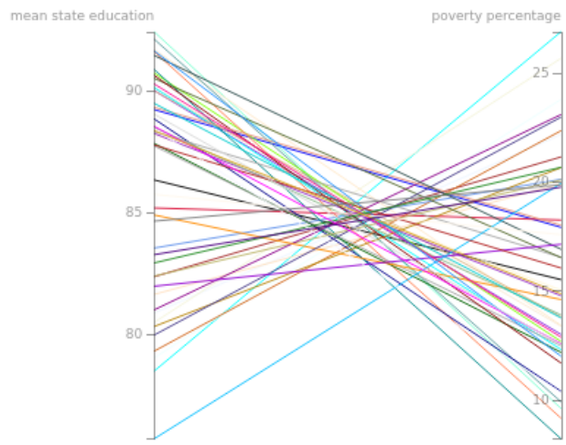


Figure 16: Rapporto tra tasso di scolarizzazione e tasso di povertà per ogni stato.

# 5 Tempi di esecuzione in locale e su cluster

In questa sezione parleremo dei tempi di esecuzione ottenuti dopo aver realizzato ogni task. Prima di parlare dei tempi però, dobbiamo descrivere sia la macchina locale, che i tipi di cluster utilizzati. Per lo svolgimento di questo progetto si è deciso di realizzare il deploy su due tipi di cluster diversi di EC2, per realizzare un confronto e determinare quale si adatta meglio all'ambiente di Spark.

Per quanto riguarda la macchina locale, i job sono stati eseguiti su un Dell XPS con le seguenti caratteristiche:

| | Processore | Cores | Memoria (GiB) | Tipo di Memoria |
|---|---|---|---|---|
| **Dell XPS** | Intel® Core™ i7-10510U | 4 | 16 | LPDDR3 |

Questo sistema però è limitato, visto che è stato eseguito su una macchina vituale che non condivide a pieno le risorse. Per esempio, la memoria RAM utilizzata è stata limitata a 10 GiB mentre il numero di processori utilizzati è stato limitato da 8 a 4.

Per quanto riguarda i cluster invece, le istanze sono state reperite dal servizio di Amazon Web Services EC2. Tramite il servizio di Elastic Map Reduce si possono richiedere istanze di EC2 che hanno già pre-installato gli ambienti Hadoop Map Reduce e Apache Spark.

La prima istanza utilizzata presa da EC2 è un istanza del tipo C5. Le istanze C5 di Amazon EC2 forniscono prestazioni elevate a costi ridotti con un rapporto ottimale tra prezzi e potenza di calcolo, per eseguire carichi di lavoro avanzati a utilizzo intensivo di calcolo. La caratteristica principale di questa famiglia di istanze è che sono state disegnate e ottimizzate per carichi di lavoro intensivi. Questo vuol dire che è destinata a sistemi che lavorano utilizzando quasi il 100% della CPU.

L'altra istanza utilizzata nel progetto è della famiglia M5. Le istanze M5 offrono un equilibro di risorse di calcolo, memoria e rete per un'ampia gamma di carichi di lavoro. In generale, si parla della scelta più flessibile ed equilibrata che si può realizzare.

| | **Processore** | **vCPU** | **Memoria (GiB)** |
|---|---|---|---|
| **c5.xlarge** | Intel Xeon Platinum serie 8000 fino a 3,6 GHz | 4 | 8 |
| **m5.xlarge** | Intel Xeon Platinum 8175M fino a 3,1 GHz | 4 | 16 |

In generale il dettaglio chiave nel confronto di queste due è il fatto che le istanze C5 costano di meno rispetto alle M5. Il confronto, per quanto riguarda le istanze xlarge non è di rilevanza visto che essendo istanze molto contenute dal punto di vista di infrastruttura, la variazione dei costi è molto piccola. Il confronto lo faremo quindi con il tipo di istanza più grande che hanno.

| | **m5.24xlarge** | **c5.24xlarge** |
|---|---|---|
| Costo | 4,608 USD/h | 4,08 USD/h |

Se supponiamo di avere queste macchine in utilizzo per un ora al giorno tutti i giorni, per un intero anno, il costo di queste sarebbe:

| | **m5.24xlarge** | **c5.24xlarge** |
|---|---|---|
| Costo | 1682 USD | 1489 USD |

Ciò vorrebbe dire che con l'utilizzo dell'istanza C5 si avrebbe un risparmio del 11,5%. A seguito si mostrerà una tabella e dei grafici con i tempi di esecuzione ottenuti per ogni task su ogni macchina.

| | **Local** | **Cluster C5** | **Cluster M5** |
|---|---|---|---|
| *Task 1* | 3759 ms | 1517 ms | 1759 ms |
| *Task 2* | 3639 ms | 1530 ms | 1697 ms |
| *Task 3* | 3236 ms | 1371 ms | 1682 ms |
| *Task 4* | 3319 ms | 1420 ms | 1589 ms |
| *Task 5* | 4658 ms | 2253 ms | 1986 ms |
| *Task 6* | 4214 ms | 1362 ms | 1544 ms |
| *Task 7* | 3587 ms | 1333 ms | 1675 ms |
| *Task 8* | 3729 ms | 1416 ms | 2000 ms |
| *Task 9* | 4166 ms | 1582 ms | 1993 ms |
| *Task 10* | 3792 ms | 1378 ms | 1534 ms |
| *Task 11* | 3593 ms | 1514 ms | 1869 ms |
| *Task 12* | 3015 ms | 1477 ms | 1552 ms |
| *Task 13* | 4642 ms | 1435 ms | 1501 ms |
| *Task 14* | 6229 ms | 2267 ms | 2473 ms |
| *Task 15* | 4048 ms | 2086 ms | 1536 ms |
| *ML Model* | 29910 ms | 17179 ms | 19390 ms |

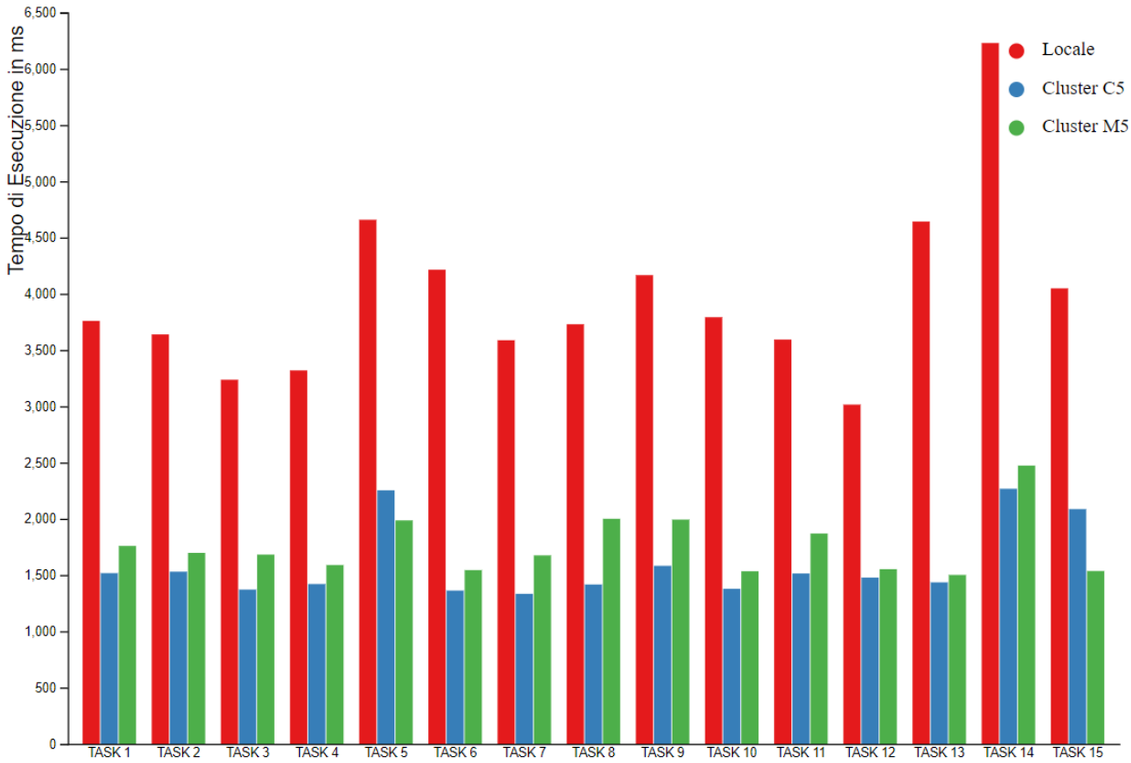


Figure 17: Tempi di Esecuzione dei Task.

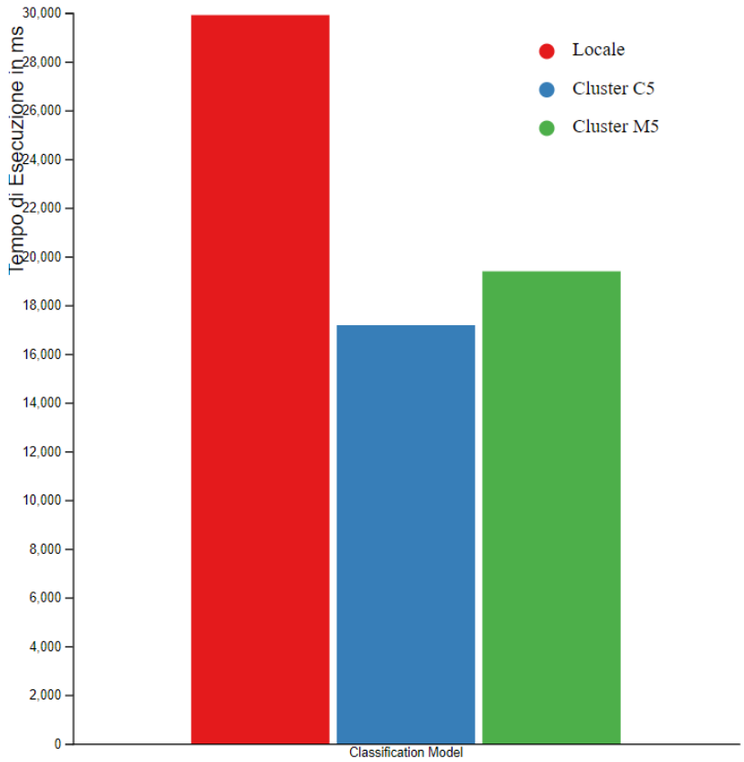


Figure 18: Tempi di Esecuzione del modello di Classificazione.

# 6 Conclusioni

Per quanto lo svolgimento dei task i risultati ottenuti ci hanno permesso di fare delle riflessioni riguardanti la quantità degli atti criminosi distribuiti nei vari stati con annesse riflessioni sui soggetti più colpiti in base ad etnia e genere.

Per il modello di classificazione, il risultato ottenuto dal punto di visto pratico è stato scarso, data la bassa precisione del modello ottenuto. Questo risultato era però abbastanza predicibile data la mancanza di dati e la mancanza di qualche feature di rilevanza in più, come lo è stato *Age*. Però l'obiettivo perseguito in questo ambito era quello di utilizzare tecniche di Machine Learning insieme allo strumento di batch processing Spark, e da questo punto di vista i risultati sono postivi. Spark si è dimostrato un ottimo strumento di appoggio per effettuare le analisi necessarie sul dataset prima di applicare il modello di classificazione. In particolare, per quanto riguarda la Feature Selection, con Spark siamo riusciti ad escludere tutte le feature che non sarebbero state importanti nel modello, e abbiamo subito trovato le feature di rilevanza che sarebbero state la chiave del modello.

Infine nello svolgimento del deploy su AWS possiamo concludere che, come ci si aspettava lo svolgimento su cluster risulta essere molto più veloce e grazie a un rapido sguardo ai grafici di confronto si nota come il cluster impiega la metà o meno della metà dei tempi di esecuzione in locale. Invece per quanto riguarda il confronto tra le due famiglie di cluster, si è visto che la famiglia di istanze a calcolo ottimizzato C5 si adatti meglio all'ambiente di Map Reduce e Spark rispetto alla famiglia di istanze general purpose M5. Scegliendo questo tipo di istanze si ottengono tempi più ridotti a costi minori, rendendolo la scelta ideale.