Università degli Studi di Modena e Reggio Emilia

Dipartimento di Ingegneria Enzo Ferrari
Laurea in Ingegneria Informatica

Studio e sperimentazione di un algoritmo per l'Entity Matching basato sulle espressioni regolari

Candidato:
Lisa Trigiante

Relatore:
Prof. Sonia Bergamaschi

Correlatori:
Dott. Giovanni Simonini
Dott. Luca Zecchini

A.A. 2019/2020

Indice

Capitolo 1

Introduzione

L'*Entity Resolution* è una tecnologia chiave per integrare semanticamente dati eterogenei. Questa tecnica ha come obbiettivo l'individuazione di tuple equivalenti (duplicate) o semanticamente equivalenti all'interno di una o più sorgenti di dati e la risoluzione di tali tuple in un unico record che ne racchiuda il contenuto informativo, risolvendo conflitti e ambiguità. Il seguente elaborato descrive tale problema e presenta un algoritmo per la sua risoluzione basato sull'utilizzo delle espressioni regolari. Da qui in poi è costituito di ulteriori 3 capitoli:

- `Capitolo secondo:` contiene una descrizione del problema di *entity resolution*, comprensivo delle tecniche che rappresentano lo stato dell'arte in tale ambito e delle metriche per la valutazione dei risultati ottenuti dai differenti metodi proposti.

- `Capitolo terzo:` descrive un problema concreto di *entity resolution* ed un algoritmo, che ho studiato e sperimentato durante la stesura di questo elaborato, che ha prodotto ottimi risultati nel contesto trattato.

- `Capitolo quarto:` conclude l'elaborato con una valutazione ed una riflessione sull'approccio precedentemente descritto, mettendone in luce i punti di forza e le criticità.

Capitolo 2

Entity Resolution

La *riconciliazione di entità*, nota generalmente con il termine inglese di *entity resolution (ER)*, consiste nell'estrazione, corrispondenza e risoluzione delle istanze di entità contenute in dati strutturati e non strutturati. *Record Linkage, Deduplication, Data Matching, Reference Reconciliation* sono solo alcuni degli altri termini usati per riferirsi a questo stesso problema, essendo sinonimi utilizzati per indicare il processo di identificazione all'interno di un insieme di dati (*dataset*) delle differenti rappresentazioni della stessa entità del mondo reale. La pluralità di nomenclatura suggerisce la numerosità degli studi effettuati nel corso degli anni in materia. I primi studi risalgono agli anni '50, tuttavia è nei tempi moderni che il tema ha acquisito più rilevanza, grazie all'evoluzione delle capacità computazionali e delle tecniche per gestire grandi quantità di dati, derivate dall'avvento dei Big Data. La preponderanza dei megadati, una raccolta di dati informativi talmente estesa in termini di volume, velocità e varietà da richiedere tecnologie e metodi analitici specifici per l'estrazione di valori e conoscenze, ha complicato in modo significativo l'entità del problema da affrontare. L'*entity resolution* si pone come obbiettivo la risoluzione delle corrispondenze fra record provenienti da una o più sorgenti che rappresentano la stessa entità del mondo reale. Ciò comporta una complessità quadratica, poichè richiede che ogni record sia confrontato con tutti gli altri: ad un aumento dei record corrisponde una crescita quadratica dei possibili confronti, che diventa computazionalmente insostenibile nel contesto dei Big Data, dove si hanno milioni di record. Un altro fattore che può determinare la crescita della complessità del nostro problema è il formato, la struttura e l'origine dei dati. L'*entity resolution* riveste un ruolo fondamentale nell'ambito della *Data Integration*, disciplina che studia i metodi per l'integrazione di più fonti di dati in una unica coerente e completa. Tale problematica è molto frequente in ambiti industriali, ove riveste un ruolo chiave la disponibilità di un data warehouse organizzato. Ad esem-

pio, un azienda strutturata in più filiali, ognuna delle quali suddivisa in più reparti proprietari di un dataset locale, avrà la necessità di integrare queste numerose sorgenti per garantire un insieme di dati coerenti a livello globale. Uno dei problemi da affrontare per la realizzazione di tale obbiettivo è lo *schema alignment*: nel caso in cui i dati provengono da una sola sorgente sono sempre allineati, mentre nel caso di più sorgenti lo schema di relazione può differire. Si definiscono non allineati due schemi che non sono composti dagli stessi attributi, in questi casi si possono usare tecniche di allineamento, come l'estrapolazione di un sottoinsieme di attributi contenenti informazioni sufficientemente simili. Distinguiamo diverse tipologie di problemi di *entity resolution* in base alla struttura dei dati:

- *Dati strutturati (clean structured dataset)*: classifichiamo un set di dati come strutturato quando i suoi record seguono lo stesso schema di attributi A1, . . . ,AN e inoltre sono relativamente puliti. I valori degli attributi sono allineati correttamente ed i campi di ogni record contengono informazioni associate solo all'attributo che li descrive. Un esempio è riportato in Tabella 2.1.

NOME	**COGNOME**	**MAIL**
Lisa	Trigiante	lisa98@live.it
Mario	Rossi	mariorossi@gmail.com

Tabella 2.1: Dati Strutturati

- *Dati strutturati sporchi (dirty structured dataset)*: classifichiamo un set di dati come strutturato sporco quando i suoi record sono strutturati con lo stesso schema di attributi A1, . . . ,AN. Tuttavia in diversi record sono presenti errori, ad esempio la presenza di valori non associati all'attributo appropriato nello schema oppure la presenza di errori di data entry. Si veda di seguito la Tabella 2.2.

NOME	**COGNOME**	**MAIL**
Mario Rossi		mariorossi@gmail.com
Mario	Rosi	mariorossi@gmail.com

Tabella 2.2: Dati Strutturati Sporchi

- *Dati non strutturati*: Sono dati che non fanno riferimento a nessuna relazione o non seguono specifici schemi, quindi nel caso di descrizioni testuali non sono altro che un semplice corpo di testo, per esempio una descrizione.

2.1 Fasi principali

Al fine di gestire la difficoltà computazionale il processo di *entity resolution* viene tipicamente suddiviso in fasi: la fase di *blocking*, la fase di *matching*, la fase di *resolution* e la fase di *clustering*.

La fase di blocking garantisce la scalabilità del processo di *entity resolution* per dataset molto voluminosi ed evita lo spreco di capacità computazionali. Il blocking è una tecnica usata per ridurre la complessità quadratica richiesta dal confronto di tutte le possibili coppie di elementi, generate eseguendo il prodotto cartesiano sull'intero dataset, consiste infatti nella suddivisione dei dati in blocchi di record di dimensioni ridotte, consentendo di effettuare il prodotto cartesiano solo sugli elementi che si trovano nello stesso blocco. I criteri di scelta usati per la suddivisione dei dati in blocchi sono definiti da una una *funzione di blocking* che valuta determinate caratteristiche e valori per includere nei blocchi solo i dati effettivamente rilevanti per il confronto a coppie.

La fase di matching è il passaggio fondamentale della risoluzione di entità, tale fase è infatti atta a garantire la semantica dell'*entity resolution* e consiste nell'identificazione delle coppie di tuple che si riferiscono alla stessa entità del mondo reale. Su ognuna delle coppie candidate, identificate dalla fase di blocking, viene applicata una *funzione di matching*, ovvero una funzione binaria che indica se i due elementi considerati si riferiscono alla stessa entità o meno. Questo passaggio implica la determinazione dei criteri per prendere la decisione locale sul risultato del confronto di due tuple di valori. Tale decisione può essere regolata da diversi approcci:

- RULE-BASED APPROACH Questo approccio sfrutta la conoscenza del dominio dei dati per prendere le decisioni locali. Il vantaggio di questo approccio è che il criterio può essere adattato alle criticità del dataset per affrontare scenari di corrispondenza complessi. Tuttavia uno svantaggio fondamentale è la necessità di una notevole conoscenza del dominio, nonché dei dati, per formulare il criterio di corrispondenza delle coppie, rendendo inefficace questo approccio quando i record contengono errori.

- CLASSIFICATION-BASED APPROACH Questo approccio si basa sull'allenamento (*training*) di un classificatore, utilizzando esempi formativi positivi e negativi. Il classificatore decide se una coppia di record è una `corrispondenza`, una `mancata corrispondenza` o una `possibile corrispondenza`, nel qual caso la decisione locale viene rivolta a un essere umano. Questo approccio basato su un classificatore e sulle tecniche di machine learning ha il vantaggio di non richiedere una

conoscenza significativa del dominio dei dati ma soltanto la conoscenza relativa agli esempi rilevanti per il classificatore, ovvero se una coppia di record contenuta in tali esempi si riferisce alla stessa entità del mondo reale o meno. Lo svantaggio di tale approccio è che spesso richiede la rilevazione di un gran numero di casi significativi per un adeguato training del classificatore.

- DISTANCE-BASED APPROACH Questo approccio si basa sull'utilizzo di metriche per il calcolo della distanza al fine di valutare la similarità degli attributi corrispondenti della coppia di tuple considerata (ad esempio calcolo della distanza euclidea nel caso di attributi numerici). La distanza tra i record è calcolata come somma ponderata delle singole distanze dei rispettivi attributi della coppia, tale distanza viene utilizzata insieme a delle soglie di minimo e di massimo per definire corrispondenze, mancate corrispondenze e possibili corrispondenze. Uno dei principali vantaggi di questo approccio è la ridotta necessità di conoscenza del dominio, limitata alla formulazione di metriche per il calcolo della distanza sugli attributi atomici, che possono essere potenzialmente riutilizzate per una grande varietà di domini di entità. Uno svantaggio di questo approccio è che spesso richiede un attento studio per la definizione dei parametri (ad esempio, quali dovrebbero essere i pesi sui singoli attributi per la somma ponderata o quali dovrebbero essere i valori delle soglie), anche se gli approcci di machine learning possono essere utilizzati inizialmente per la regolazione di tali parametri.

La fase di *resolution (data fusion)* si occupa di produrre un record rappresentativo per una data entità del mondo reale. Una volta individuati tutti gli elementi che si riferiscono a tale entità, si applica su di essi una *funzione di risoluzione* che utilizza una *funzione di aggregazione* specifica per ciascun attributo, permettendo di generare un unico elemento rappresentativo dell'entità stessa e risolvendo i conflitti e le ambiguità presenti negli elementi originari. Prendiamo come esempio la Tabella 2.3 che contiene tre record che si riferiscono alla stessa entità.

ID	**MARCA**	**MODELLO**	**SCHERMO**
www.ebay.com//143	Dell	1907FP	17
www.ohc24.ch//67	del	1907FPv	19
www.pc-canada.com//253	Dell	1907FPt	19

Tabella 2.3: Record che descrivono la stessa entità

Per ottenere un unico record rappresentativo dell'entità a partire da tali tuple usiamo una funzione di aggregazione specifica per ogni attributo: per quanto riguarda l'attributo `ID` usiamo la funzione `RANDOM` che sceglie il valore di uno qualsiasi tra gli identificatori dei record originali; per l'attributo `MARCA` usiamo la funzione `VOTE` che sceglie il valore più ricorrente dell'attributo, in questo caso il valore prescelto è *Dell*; per l'attributo `MODELLO` scegliamo la funzione `MIN` che prende il valore minore tra gli attributi (trattandosi di dati testuali si basa sull'ordinamento alfabetico); per l'attributo `SCHERMO` al contrario scegliamo la funzione `MAX`. Il record ottenuto dalla risoluzione dell tuple iniziali è riportato in Tabella 2.4.

ID	MARCA	MODELLO	SCHERMO
www.ebay.com//143	Dell	1907FP	19

Tabella 2.4: Record rappresentativo dell'entità

La fase di *clustering* garantisce che le decisioni sulla risoluzione dei record in entità abbiano non solo una consistenza locale, ma che siano coerenti anche da un punto di vista globale. Lo scopo del clustering è quello di raggiungere una decisione coerente a livello globale su come partizionare il set di record in modo che ogni partizione (*cluster*) faccia riferimento a un entità distinta e che le partizioni differenti si riferiscano a entità differenti. Un possibile metodo per l'esecuzione del clustering è l'applicazione della chiusura transitiva sull'intero set di dati, che viene considerato come un grafo in cui ogni nodo rappresenta un record e ogni arco la funzione di corrispondenza tra due record. Esiste una grande varietà di algoritmi di clustering per la risoluzione di entità, questi tendono a non vincolare il numero di cluster nell'output poiché il numero di entità nel set di dati non è in genere noto a priori. Una delle strategie di clustering più semplici consiste nella suddivisione dell grafo tramite una singola scansione degli archi che ne connettono gli elementi. Essenzialmente, questa strategia pone un'elevata fiducia nelle decisioni locali di corrispondenza: anche un paio di corrispondenze errate possono modificare in modo significativo i risultati della risoluzione di entità. All'altro estremo, un algoritmo di clustering robusto ma costoso è il clustering di correlazione. L'obiettivo di questa tecnica è quello di trovare una partizione dei nodi del grafo che riduce al minimo i disaccordi tra il clustering e le decisioni locali: per ogni coppia di nodi nello stesso cluster che non sono connessi da un arco viene assegnata una penalità di correlazione di 1, per ogni coppia di nodi in cluster diversi connessi da un arco viene assegnata una penalità di correlazione di 1. Il clustering di correlazione cerca di calcolare il clustering che minimizza la somma complessiva delle penalità (cioè i disaccordi).

2.2 Tecniche di valutazione

Effettuare la valutazione delle tecniche di *entity resolution* richiede di introdurre delle definizioni per poter saggiare la correttezza dei risultati ottenuti dal confronto dei record. A tale scopo introduciamo i concetti di *vero positivo*, *vero negativo*, *falso positivo* e *falso negativo*.

- *Vero positivo*: coppia di record correttamente classificata come rappresentativa della stessa entità del mondo reale. Ad esempio un record A ed un record B che rappresentano lo stesso oggetto del mondo reale ed il cui confronto produce esito positivo.

- *Falso positivo*: coppia di record classificata in modo erroneo come rappresentativa della stessa entità del mondo reale. Ad esempio un record A ed un record C che non rappresentano lo stesso oggetto del mondo reale tuttavia il loro confronto produce esito positivo.

- *Vero negativo*: coppia di record correttamente classificata come non rappresentativa della stessa entità del mondo reale. Ad esempio un record A ed un record C che non rappresentano lo stesso oggetto del mondo reale ed il loro confronto produce esito negativo.

- *Falso negativo*: coppia di record classificata in modo erroneo come non rappresentativa della stessa entità del mondo reale. Ad esempio un record A ed un record B che rappresentano lo stesso oggetto del mondo reale tuttavia il loro confronto produce esito negativo.

		Predicted	
		Negative	**Positive**
Actual	**Negative**	True Negative	False Positive
	Positive	False Negative	True Positive

Figura 2.1: Confusion Matrix

Partendo da questi concetti introduciamo le due metriche usate per effettuare la misurazione delle prestazioni di *entity resolution*, esse sono due comuni classificazioni statistiche: precisione (*precision*) e recupero (*recall*). La precision può essere vista come una misura di esattezza o fedeltà, mentre la recall può essere vista come una misura di completezza.

- *Precision*: Chiamata anche *confidence*, misura la percentuale dei veri correttamente classificati rispetto a tutti i classificati come veri.

$$\text{Precision} = \frac{True\ Positive}{True\ Positive + False\ Positive}$$

Figura 2.2: Formula per calcolare Precision

- *Recall* Chiamata anche *sensitivity*, misura la proporzione dei predetti veri correttamente classificati rispetto a tutti i veri.

$$\text{Recall} = \frac{True\ Positive}{True\ Positive + False\ Negative}$$

$$= \frac{True\ Positive}{Total\ Actual\ Positive}$$

Figura 2.3: Formula per calcolare Recall

Una tecnica di *entity resolution* risulta essere migliore di un altra se la coppia di valori precision e recall è più vicina al 100%. Vi è però una difficoltà nel cercare di comprendere quali coppie di precision e recall siano migliori rispetto ad altre. Ad esempio (92.7%, 94.4%) e (93.8%, 93.4%) sono coppie i cui valori sono molto vicini e quindi determinare la coppia migliore può risultare complicato. A tale scopo introduciamo una nuova metrica atta a confrontare queste coppie di valori per determinarne la migliore in termini prestazoinali: *F1 score*. Calcolata come media armonica di precision e recall, avendo entrambe i veri positivi al numeratore si confrontano i reciproci per avere un confronto omogeneo.

$$F1 = 2 \times \frac{Precision * Recall}{Precision + Recall}$$

Figura 2.4: Formula per calcolare F1 Score

Questo valore ci permette di confrontare diverse coppie di precision e recall. Prendendo come esempio quello precedente otteniamo rispettivamente due valori di *F1 score*: 93.54% e 93.60% e quindi con una differenza di 0.06% possiamo affermare che il secondo risultato è lievemente migliore. Il grafico riportato di seguito mostra i valori assunti da F1 Score in relazione alla variazione dei valori assunti da precision e recall.

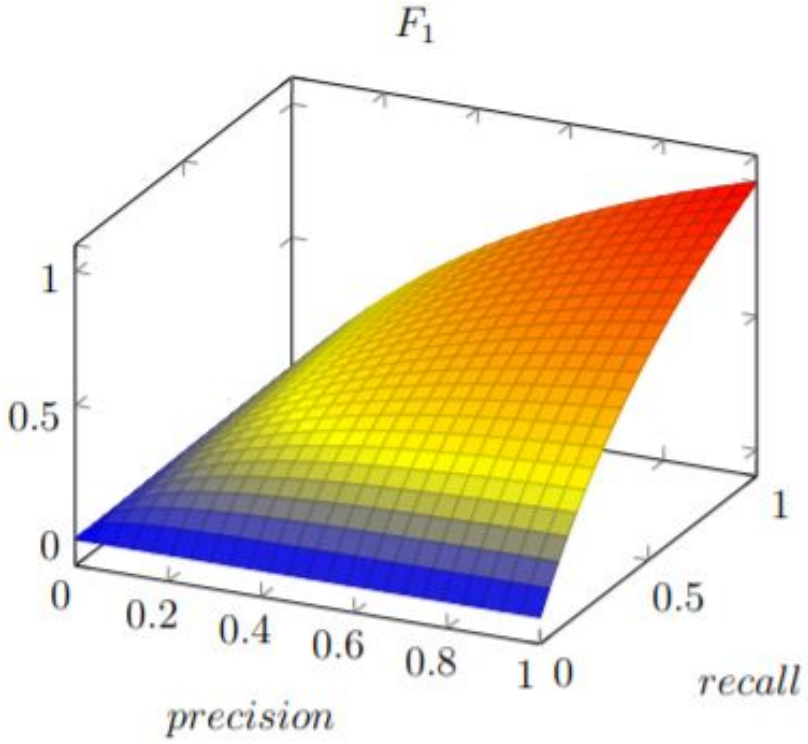

Figura 2.5: F1 Score al variare di precision e recall

2.3 Stato dell'arte

Allo stato attuale, gli algoritmi più performanti per affrontare il problema dell'*entity resolution*, in particolare dell'*entity matching*, sono basati sulle conoscenze di machine learning e deep learning. Il machine learning è la branca dell'intelligenza artificiale che studia gli algoritmi di apprendimento automatico di una macchina. Comprende varie tecniche, casi e approcci. In generale gli algoritmi di machine learning analizzano un insieme di dati per costruire un modello previsionale in grado di classificare gli oggetti in autonomia e rispondere correttamente alle domande su un particolare dominio della conoscenza. Il deep learning è un caso particolare del machine learning che si basa sull'utilizzo delle reti neurali artificiali (*neural network*), per elaborare le informazioni in modo non lineare. Una rete neurale artificiale è un modello matematico che si ispira al funzionamento delle reti neuronali biologiche. Nel cervello umano i neuroni sono interconnessi dalle sinapsi e ci consentono di ragionare e comandare ogni funzione e nervo del nostro corpo. In campo informatico la rete neurale è abbastanza simile, è composta da nodi (neuroni) e da archi (sinapsi) che collegano i nodi. Per affrontare ciò che riguarda l'applicazione dei metodi di machine learning e deep learning al problema di entity matching introdurremo due librerie Python che utilizzano rispettivamente metodi di machine learning (Magellan) e deep learning (DeepMatcher) e rappresentano lo stato dell'arte nei rispettivi ambiti.

- *Magellan*: sviluppato dall'AnHai Group presso l'Università del Wisconsin, Magellan è uno strumento di entity matching che consente di abbinare due tabelle utilizzando tecniche supervisionate di apprendimento. In particolare, si tratta di un sistema che copre l'intera pipeline di entity matching. Magellan si basa su tre pacchetti `Python` di tipo open source:
  1) *py_stringmatching*, che implementa funzioni di string tokenizers e string similarity;
  2) *py_stringsimjoin*, che sfrutta il pacchetto precedente per trovare tutte le coppie di stringhe corrispondenti provenienti da due set di stringhe;
  3) *py_entitymatching*, che utilizza entrambi i pacchetti precedenti per trovare tutte le coppie di tuple corrispondenti provenienti da due tabelle;
  Infatti, Magellan permette di eseguire l'entity matching grazie all'utlizzo di alcune tra le più usate tecniche di machine learning per il riconoscimento delle corrispondenze: *decision tree*, *random forest*, *SVM*, *naïve Bayes*, *linear regression* e *logistic regression*. Inoltre l'utente può definire delle proprie regole per il riconoscimento delle corrispondenze.

- *DeepMatcher*: esegue la corrispondenza di entità o elementi testuali tramite l'uso di tecniche di deep learning, usando perciò reti neurali integrate. I possibili modelli di rete neurale, che possono essere personalizzati dagli utenti, sono:

  1) *SIF (Smooth Inverse Frequency)*, che determina una corrispondenza o una mancata corrispondenza considerando le parole presenti in ogni coppia di valori di un attributo, senza preoccuparsi del loro ordine;
  2) *RNN (Recurrent Neural Network)*, che considera le sequenze di parole;
  3) *Attention*, che considera l'allineamento delle parole, senza preoccuparsi del loro ordine;
  4) *Hybrid*, che considera l'allineamento delle sequenze di parole, selezionate come modello predefinito.

  Parallelamente a questa situazione la ricerca accademica è molto attiva sul tema, in particolare l'Università degli studi di Modena e Reggio Emilia sta affrontando questa tematica attraverso il DBGroup guidato dalla docente e ricercatrice Sonia Bergamaschi. [1, 2, 3, 4]

Capitolo 3

Algoritmo

3.1 Contesto

Il percorso di attività progettuale su cui si basa la presente tesi ha origine dall'algoritmo realizzato da Luca Zecchini, studente magistrale dell'Università degli studi di Modena e Reggio Emilia, nell'ambito della competizione ACM SIGMOD Programming Contest 2020. Il cui obiettivo era la realizzazione dell'entity matching su un dataset labellato contenente le specifiche di 29.787 annunci relativi a fotocamere provenienti da 24 differenti siti di e-commerce. La valutazione delle prestazioni della soluzione proposta era effettuata in base al valore di F-score raggiunto su un evaluation set segreto.

Il primo approccio per risolvere il problema era basato sull'uso delle due librerie Python: Magellan e DeepMatcher, che si sono dimostrate in grado di raggiungere ottimi risultati sul dataset (in particolare il modello RNN di DeepMatcher), con un risultato di F-score pari a 0,98 se applicato sulle prime 4 parole del titolo dell'annuncio (unico attributo sempre presente e contenente le sole informazioni essenziali per l'identificazione). Spostandosi però sull'intero dataset, il valore di F-score è crollato a 0,47, a causa di un numero abnorme di falsi positivi, tale da determinare una precision molto bassa (0,32). La causa di questo fenomeno è da ricercare nella natura dei dati stessi: nel dataset considerato la determinazione dei match si basa su piccole variazioni nel nome del modello (spesso di una singola lettera o cifra) dipendenti dalla relativa marca. Per questo motivo, i modelli appresi studiando i dati etichettati (pochi e polarizzati solo su alcune marche e modelli) non sono generalizzabili, e non si possono estendere indiscriminatamente all'intero dataset.

A fronte di ciò è stato deciso di abbandonare l'approccio basato sul machine learning per cercare una soluzione alternativa. Siccome per l'identi-

ficazione dei prodotti ci si può basare unicamente su marca e modello, e tali elementi sono solitamente presenti nel titolo dell'annuncio, la soluzione finale si basava sulla ricerca di questi due elementi in tale attributo: la marca utilizzando una lista di marche comuni, il modello grazie all'uso di una espressione regolare in grado di individuare le stringhe composte sia da lettere che da cifre (con diverse integrazioni per gestire le varie eccezioni possibili). La definizione di tutte le operazioni relative ad ogni singola marca per la pulizia dei modelli comprende ad esempio la gestione di prefissi e suffissi o delle equivalenze tra le diverse grafie (anche nel caso di errori di battitura) e tra i vari alias usati per indicare lo stesso modello. L'implementazione di queste funzionalità ha richiesto moltissimo lavoro umano, dovendo effettuare manualmente lo studio dei dati e la ricerca dei pattern tipici di ogni marca. Una volta individuata la coppia marca-modello, il passaggio successivo consiste nella realizzazione di un inverted index sulla combinazione di questi due elementi, andando a definire gli insiemi di match con chiusura transitiva intrinseca. La soluzione fornita da questo approccio è stata in grado di raggiungere una F-score di 0,99 sull'evaluation set, con precision 0,99 e recall 0,98, risultato molto migliore rispetto a quelli precedentemente ottenuti con i metodi di machine learning e deep learning.

3.2 Descrizione

Partendo da queste valutazioni il mio lavoro è stato quello di adattare l'algoritmo presentato in precedenza ad un nuovo dataset, fornito dagli organizzatori di ACM SIGMOD Programming Contest 2020 nell'ambito del workshop DI2KG 2020. Tale set di dati è composto da 16.662 specifiche di prodotto, ognuna di tali specifiche si riferisce ad un monitor, o in alcuni casi a un accessorio (ad esempio una custodia) e sono tutte memorizzate in un file dedicato. I file sono distribuiti in 26 cartelle, ognuna delle quali rappresenta il sito di e-commerce da cui vengono estratti (ad esempio www.ebay.com). Come illustrato nella Tabella 3.1, la distribuzione delle specifiche tra le fonti non è uniforme, con poche cartelle (in particolare la fonte citata in precedenza) contenenti la maggior parte dei file ed altre contenti un numero minore di record.

SOURCE	**SPECS**	**SOURCE**	**SPECS**
ca.pcpartpicker.com	712	www.mediashopuk.com	176
catalog.com	180	www.mrhightech.com	176
ce.yikus.com	257	www.nexus-t.co.uk	136
www.best-deal-items.com	2,801	www.odsi.co.uk	199
www.cleverboxes.com	469	www.officedepot.com	235
www.ebay.com	4,281	www.ohc24.ch	798
www.getprice.com.au	268	www.pc-canada.com	120
www.hardware-planet.it	305	www.pcconnection.com	623
www.imldirect.it	763	www.planet-computer.it	190
www.itenergy.co.uk	366	www.shopmania.com	108
www.jrlinton.co.uk	1,089	www.softwarecity.ca	401
www.kingsfieldcomputers.co.uk	571	www.vology.com	528
www.makingbuyingeasy.co.uk	603	www.xpcpro.com	307

Tabella 3.1: Number of specifications extracted from each source

Ogni specifica è archiviata nel proprio file JSON, contenente un dizionario composto da coppie <*attribute name*: *attribute value*> che rappresentano le caratteristiche del prodotto. Un esempio della struttura della specifica di un prodotto, in questo caso il file 9330.json, tratto dalla cartella www.ebay.com, può essere visto di seguito. Ovviamente, il numero utilizzato come nome del file rappresenta l'identificatore univoco del prodotto considerato all'interno della sua cartella.

```
{
    "<page title>": "Dell E190S 19 inch Flat Panel Screen LCD
        Monitor EE468014 Very Good Monitors 5397063039388 | eBay",
    "aspect ratio": "4:3",
    "brand": "Dell",
    "contrast ratio": "800",
    "display technology": "CRT",
    "max. resolution": "1280 x 1024",
    "model": "E190S",
    "mpn": "E190S, F779N",
    "response time": "5 ms",
    "screen size": "19\"",
    "upc": "884116063292"
}
```


È possibile individuare immediatamente un attributo indicato come <page title>. Questo attributo è particolarmente importante perché è l'unico che è presente in ogni specifica e in genere contiene elementi significativi come il marchio (*Dell*) e il modello (*E190S*) del computer a cui fa riferimento. Al contrario, gli altri attributi sono molto vari. È possibile osservare che ci sono molti attributi diversi ma nessuno di essi, escludendo <page title>, è presente in tutte le 26 fonti. Guardiamo ad esempio le differenze tra il file JSON precedente e quello riportato di seguito (file 11.json tratto dalla cartella www.kingsfieldcomputers.co.uk).


```
{
    "<page title>": "FWD-S42H1, 1625962, Sony, LCD Monitor,
       Kingsfield Computer Products Ltd",
    "aspect ratio": "16:9",
    "brand name": "Sony",
    "brightness": "700 cd/m\u00b2",
    "builtin devices": "Speaker",
    "colour support": "1.06 Billion Colors",
    "composite video": "Yes",
    "contrast ratio": "1,000:1",
    "depth": "124.5 mm",
    "dvi": "Yes",
    "hdcp supported": "Yes",
    "height": "577.9 mm",
    "horizontal viewing angle": "178\u00b0",
    "manufacturer": "Sony Corporation",
```

```
    "manufacturer part number": "FWD-S42H1",
    "maximum resolution": "1920 x 1080",
    "mount type": [
        "Arm-mountable",
        "Wall Mountable"
    ],
    "number of screens": "1",
    "number of speakers": "2",
    "product model": "FWD-S42H1",
    "product name": "FWD-S42H1 Widescreen LCD Monitor",
    "product series": "FWD",
    "product type": "LCD Monitor",
    "response time": "8 ms",
    "screen mode": "Full HD",
    "screen size": "106.7 cm (42\")",
    "speakers": "Yes",
    "svideo": "Yes",
    "vertical viewing angle": "178\u00b0",
    "vga": "Yes",
    "weight approximate": "29.94 kg",
    "width": "984.3 mm"
}
```


C'è una grande varietà anche tra le specifiche contenute nella stessa fonte: per esempio nella distribuzione di alcuni attributi all'interno della cartella `www.ohc24.ch` (si vedano di seguito i file 0.json e 878.json). In questo caso, alcuni altri attributi oltre a <`page title`> sono presenti in ogni file, ma in molti casi questo non accade.


```
{
    "<page title>": "OHC24 Shop : Monitor > Monitor search help >
       Epson A61B133111",
    "brightness": "690 cd/m2",
    "colour": "Cool white",
    "compliant standards": "FCC Class A certified, CSA, UL, TUV,
       VCCI Class A ITE",
    "device type": "Customer display",
    "dimensions wxdxh": "16.5 cm x 5.1 cm x 6.9 cm",
    "product description": "Epson DMD110 - customer display",
    "weight": "0.6 kg"
}
```

```
{
    "<page title>": "OHC24 Shop : Monitor > Monitor search help
       > Iiyama
       XU2390HS-B1 58 > 4CM 23IN LED 58.4 cm (23\")/",
    "aspect ratio": "Widescreen - 16:9",
    "brightness": "250 cd/m2",
    "colour": "Black",
    "colour support": "16.7 million colours",
    "compliant standards": "DDC-2B, VESA DPMS, TUV Bauart,
       VCCI Class B, EAC",
    "contrast ratio": "1000:1 / 5000000:1 (dynamic)",
    "device type": "LED-backlit LCD monitor - 23\"",
    "dimensions wxdxh": "53.25 cm x 18 cm x 38.75 cm",
    "display position adjustments": "Tilt",
    "environmental standards": "ENERGY STAR Qualified",
    "input connectors": "HDMI, DVI-D, VGA",
    "native resolution": "FullHD 1920 x 1080",
    "panel type": "IPS",
    "pixel pitch": "0.265 mm",
    "product description": "Iiyama ProLite XU2390HS-1
       - LED monitor - 23\"",
    "response time": "5 ms",
    "speakers": "Integrated",
    "weight": "4 kg"
}
```


È importante notare che le fonti sono sporche, quindi è possibile (e in realtà abbastanza comune) trovare corrispondenze anche tra le specifiche provenienti dalla stessa fonte. Inoltre, viene applicata la chiusura transitiva alle corrispondenze (se A corrisponde a B e B corrisponde a C, anche A corrisponde a C).

Di seguito è riportato un esempio molto semplice di corrispondenza tra due specifiche: è possibile dire che la corrispondenza è determinata solo dall'attributo primario <page title>, che contiene la marca e il modello: in questo caso per entrambe le specifiche sono rispettivamente *HP* e *U160*, nel primo caso la specifica è pulita (non sono presenti errori di battitura, non vengono utilizzati sinonimi e il prefisso *U* è associato al resto del nome del modello, senza spazi vuoti o trattini). La seconda specifica presenta invece l'utilizzo di un sinonimo per il marchio del computer (Hewlett-Packard al posto di HP). Per quanto riguarda gli altri attributi, le due specifiche non presentano affinità interessanti, la prima contiene i due attibuti *manufacturer*

e *model name* che sono determinanti per identificare in modo univoco il monitor, mentra la seconda specifica contiene il campo *product description* con le informazioni su marca e modello del prodotto. Di seguito sono presentati come esempio i file www.officedepot.com//312 e www.ohc24.ch//437.


```
{
    "<page title>": "HP Promo U160 156 LED LCD Monitor 12 ms
       by Office Depot",
    "adjustable display angle": "Yes",
    "adjustable display height": "No",
    "adjustable display pivot": "No",
    "backlight technology": "LED",
    "brand name": "HP",
    "brightness": "180 Nit",
    "color": "Black",
    "contrast ratio": "500:1",
    "depth": "0.6\"",
    "ecolabel": "ENERGY STAR; EPEAT Silver",
    "height": "10.7\"",
    "item ": "779618",
    "manufacturer": "Hewlett-Packard",
    "manufacturer ": "QQ3782",
    "maximum resolution": "1366 x 768",
    "model name": "U160",
    "mount type": "Desk Mountable",
    "number of screens": "1",
    "operating power consumption": "5 W",
    "platform supported": "PC",
    "product line": "Promo",
    "product name": "U160 15.6-inch LED Backlit Monitor",
    "product type": "LCD Monitor",
    "response time": "12 ms",
    "screen mode": "WXGA",
    "screen size": "15.6\"",
    "screen surface": "Anti-glare",
    "speakers": "No",
    "standby power consumption": "500 mW",
    "usb": "Yes",
    "usb standard": "USB 2.0",
    "vertical viewing angle": "65\u00b0",
    "viewing angle": "90\u00b0",
```

```
    "warranty length": "3-year limited",
    "weight": "3.39 lb",
    "width": "14.8\""
}

{
    "<page title>": "OHC24 Shop : Monitor > Monitor search help >
       Hewlett-Packard U160 15.6IN USB 2.0 HP U160 D4T56AA#ABB",
    "aspect ratio": "Widescreen - 16:9",
    "brightness": "180 cd/m2",
    "colour": "Black",
    "colour support": "262,144 colours",
    "compliant standards": "Plug and Play, VCCI, C-Tick, BSMI,
       GOST, cUL, SABS, SASO, CB, CCC, PSB, FCC, C-ETL, RoHS,
       IEC 60950-1, KCC, WEEE, Green Mark, ICES, S Mark,
       TUV Bauart, KC, CECP, CEL, Nordic Ecolabel,
       GOST-R 50377-1992, NOM-019-SCFI-1998",
    "contrast ratio": "500:1",
    "device type": "LED-backlit LCD monitor - 15.6\"",
    "dimensions wxdxh": "37.7 cm x 2.6 cm x 27.85 cm - with stand",
    "display position adjustments": "Tilt",
    "environmental standards": "ENERGY STAR Qualified",
    "input connectors": "USB",
    "localisation": "English / Europe",
    "native resolution": "1366 x 768 at 60 Hz",
    "panel type": "TN",
    "pixel pitch": "0.252 mm",
    "product description": "HP U160 - LED monitor - 15.6\"",
    "response time": "12 ms",
    "screen coating": "Anti-glare",
    "weight": "1.54 kg"
}
```


Questo esempio è sufficiente per denotare il problema principale con gli attributi secondari, oltre alla distribuzione illustrata in precedenza. Spesso lo stesso contenuto è rappresentato da due attributi con nomi diversi o due attributi con lo stesso nome si riferiscono a caratteristiche diverse. In aggiunta a questo, è possibile rilevare l'eventuale presenza di alcuni dettagli sugli accessori, i quali però non contribuiscono all'identificazione del prodotto: un *HP U160* con una custodia, rappresenta lo stesso prodotto di un *HP U160*

venduto da solo. Analogo è il caso di due prodotti uguali, il cui colore è diverso: questo non li rende prodotti diversi. Ciò è particolarmente significativo, perché spesso il colore è presente come suffisso nel nome del modello. L'obiettivo del progetto è quello di produrre un file in formato *CSV* contenente tutte le coppie corrispondenti trovate nel dataset, strutturate in due colonne (`left spec id`, `right spec id`), con ogni riga che rappresenta una coppia di specifiche.

Al fine di confrontare le specifiche, si è deciso di considerare solo l'attributo speciale <`page title`>. Infatti, questo attributo è presente in ogni specifica e contiene in quasi tutti i casi i due elementi che sono sufficienti a identificare in modo univoco il monitor a cui si riferisce: il marchio e il modello. Partendo da questo presupposto, è stata modificata la struttura del set di dati, riducendola a un dataframe in formato *Pandas* composto solo dalla colonna `id` (il percorso espresso nel set di dati, che fungeva da uniqueidentifier per la specifica) e dalla colonna `page_title`. Il secondo attributo è stato convertito in minuscolo, mentre il primo era già normalizzato da questo punto di vista, e così anche tutti i nomi di attributo (tutti già in minuscolo). Ulteriori miglioramenti potrebbero essere apportati dall'uso di alias, per risolvere il problema dei sinonimi più frequenti.

Il primo step da affrontare è l'estrazione dall'attributo primario <`page title`> della marca e del modello. Per risolvere questo problema è stato osservato che il numero di marchi con una distribuzione elevata è piuttosto limitato, gestibile attraverso una lista contenete i brand più famosi o quelli più ricorrenti nei record. Mentre nella maggior parte dei casi il modello viene espresso come una stringa composta sia da lettere che da cifre, in modo da poter essere estratto utilizzando una *regular expression*. L'estrazione del marchio sarà effettuata selezionando la prima occorrenza di una corrispondenza tra una delle parole contenute in <`page title`> e la lista di marchi noti. Mentre per il modello, la prima occorrenza di una stringa contenente sia lettere che numeri (tale approccio ha prodotto un risultato corretto per numerosi record). L'obbiettivo è di ridurre l'attributo <`page title`>, nel caso in cui entrambi gli elementi siano rilevati, a una stringa composta solo da marca e modello, quindi di eseguire l'inverted index su questo nuovo attributo, determinando i blocchi di elementi corrispondenti (elementi con lo stesso marchio e lo stesso modello), da cui saranno determinate le coppie corrispondenti attraverso il prodotto cartesiano interno a ogni blocco, filtrando le coppie duplicate grazie ai principi di identità e riflessività.

3.3 Codice

Il codice è sostanzialmente strutturato in due parti: la prima lavora su ogni specifica per estrarre marca e modello, mentre la seconda esegue un *inverted index* sulle specifiche per le quali sono state trovate queste informazioni (le altre specifiche saranno trattate con una soluzione parallela), creando i blocchi e susseguentemente generando le coppie di specifiche che hanno trovato corrispondenza, salvandole nel file in formato CSV.

All'inizio due cicli nidificati leggono le specifiche (JSONfiles) contenute in ogni cartella; il contenuto di ogni file viene portato nel formato `dictionary`, in cui il nome dell'attributo viene utilizzato come chiave e il valore dell'attributo come valore. Un dizionario vuoto denominato `monitor` viene inizializzato e rappresenta la specifica corrente. Le due liste vuote `solvedspecs` e `unsolvedspecs` saranno utilizzate per separare le tuple per le quali è possibile estrarre il marchio e il modello da quelle per cui uno di questi o entrambi non sono stati rilevati.

```
# Read JSON specifications
path = 'Dataset/2013_monitor_specs'

solved_specs = []
unsolved_specs = []

for folder_name in os.listdir(path):
        for file_name in os.listdir(path + '/' + str(
           folder_name)):
                with open(path + '/' + str(folder_name) + '
                   /' + str(file_name), 'r') as  data:
                        monitor = {}

                        # Read the JSON content as a
                            dictionary
                        spec = json.load(data)
                        spec = dict((k.lower(), v) for k, v
                             in spec.items())
```

L'identificatore di ogni tupla, chiamato id, viene semplicemente ottenuto riproducendo il percorso del `JSON` file (concatenando il nome della cartella e il nome del file, utilizzando `//` al posto di `/` per separarli). Quindi, viene aggiunto alla tupla anche l' attributo `page_title`, che rappresenta la versione

normalizzata dell'attributo speciale <page title>. La normalizzazione si realizza convertendolo in minuscolo, quindi sostituendo al suo interno tutti i caratteri di punteggiatura presenti nell'elenco `punctuation` con un carattere inattivo; quindi, se due parole sono solo separate da una virgola senza uno spazio (ad esempio,"computer,monitor"), saranno correttamente separate e considerate come due parole diverse. Al contrario, il carattere `-` viene sostituito una stringa vuota, poichè viene spesso utilizzato all'interno di nomi dei modelli: questo può essere considerato come il primo passo verso la normalizzazione dei nomi dei modelli (ad esempio,"U-160"diventa"u160").

```
# String normalization: punctuation (substituted by space)
    and stop characters (just deleted)
punctuation = [",", ":", ";", "!", "?", "(", ")", "[", "]",
    "{", "}", "/", "|", '"', "*"]
stop_chars = ["-"]

# Add file path as 'id' attribute
monitor['id'] = str(folder_name) + '//' + str(file_name
    [:-5])

# Add and normalize 'page_title' attribute
monitor['page_title'] = spec['<page_title>'].lower()
for p in punctuation:
        monitor['page_title'] = monitor['page_title'].
            replace(p, ' ')
for c in stop_chars:
        monitor['page_title'] = monitor['page_title'].
            replace(c, '')
```

Per i passaggi successivi, l'attributo `page_title` non viene utilizzato come stringa, ma viene diviso ed utilizzato come elenco delle parole che lo compongono. Questo elenco di parole è denominato `splitted`. La prima operazione è il recupero e la sostituzione degli alias, memorizzati in un apposito dizionario, all'interno di `splitted`. Questa operazione risolverà il problema dei sinonimi o delle versioni errate utilizzate per indicare la stessa marca (ad esempio `"hewlett packard"` per `"hp"`).

```
splitted = monitor['page_title'].split()

# String normalization: aliases
aliases = {"panosonic": "panasonic", "ssamsung":
```

```
    "samsung", "repairsony": "sony", "vivicam": "v",
    "plus": "+", "1080p": "", "720p": "", "led": "",
    "buy": "", "series": "", "reference": "", "pos":
    "posx", "hewlett": "hp", "aversion": "aver",
    "hannspree": "hanns.g", "hanns": "hanns.g",
    "hannsg": "hanns.g", "asustek": "asus",
    "bradley": "allenbradley"}

# Resolve aliases
for s in splitted:
        if s in aliases.keys():
                index = splitted.index(s)
                splitted[index] = aliases[s]
                s = splitted[index]
```

Dopo questa prima sostituzione, per il recupero del marchio viene utilizzato il valore di `splitted`; confrontando ogni parola con una lista di marchi noti. Nel caso in cui l'attributo contenga più di un marchio (come in seguito per il modello), viene considerato solo il primo marchio rilevato. Per preparare l'estrazione del modello, per ogni marca sono definiti alcuni elementi specifici per normalizzare le versioni dei loro modelli. Questa operazione viene eseguita tramite 4 liste ed un dizionario:

- `prefixes` e `suffixes` sono due liste che contengono gli elementi ricorrenti che possono essere visualizzati prima o dopo le stringhe alfanumeriche identificate come modelli e che devono essere considerate parte del nome del modello;
- `models` è una lista che contiene parole solo alfabetiche o solo numeriche (quindi, non verrebbero rilevate senza l'uso di questo elenco) che devono essere considerate come modelli;
- `exceptions` è una lista che contiene le parole che sono sia alfabetiche che numeriche, ma che devono essere ignorate perché non rappresentano un modello;
- `equivalences` è un dizionario che consente di conformare versioni diverse del nome di un modello scegliendo uno standard.

```
# List of common manufacturers for the brand extraction
brands = [ "3m", "acer", "adata", "adesso",
"allenbradley", "amd", "aoc", "apc", "apple", "argus",
```

```
"asus", "aten", "aver", "avocent", "barco", "belkin",
"benq", "compaq",  "coolemaster", "ctl", "dahua",
"datalogic", "dell", "delta", "doublesight", "eizo",
"elite", "elo", "epson", "ergotron", "fellowes",
"fujitsu", "gateway", "hanns.g", "hyundai", "hp",
"ibm", "iiyama", "infocus", "intel", "iogear",
"itec","kingsfield", "kingston", "lacie", "leica",
"lenovo", "lexus", "lg", "lilliput", "marshall",
"mitsubishi", "motorola", "msi", "nanov", "nec",
"neovo", "newstar", "nvidia", "optiquest", "panasonic",
"peerless", "pelco", "philips", "pioneer", "planar",
"posx", "pny",  "qnix", "ricoh", "roline","samsung",
"sharp", "siemens", "sony", "startech", "sunbrite",
"targus", "toshiba", "touchsystems", "trust",  "v7",
"viewsonic", "wortmann", "yamakasi", "yiynova", "zeiss" ]

# Extract brand
brand = 'none'
for s in splitted:
        if s in brands:
                brand = s
                break

# Manage specific elements of each brand
prefixes = []
suffixes = []
models = [] # only alphabetic or only numeric models
          (so not retrieved through regular expressions)
exceptions = [] # alphanumeric words which
                  do not represent models
equivalences = {} # equivalent names for the same model
```

Di seguito è riportato un esempio della definizione di questi elementi, effettuata sul brand Dell.

```
if brand == 'dell':
        prefixes = [ 'optiplex', 'xps' ]
        suffixes = [ 'fpt', 'fpw', 'fpc']
        models = [ '3202676', '1708' '2405', '760',
           '745', '320', '1720', '3000' ]
        exceptions = [ '10x', '12v', '1.5a', '15x',
```

```
        '16v', '15in', '12in', '13in', '2x', '42w',
        '5x', '90degrees', 'a02', 'bn41', 'cj167',
        'claa170ea07', 'clkaa170ea10', 'core2',
        'core2duo', 'ddr2', 'f846n', 't6116' ]
    equivalences = {'ultrasharp1907fpc': '1907fpc'}
```

La prima operazione da effettuare è la gestione di suffissi e prefissi, attraverso le apposite liste, concatenandoli alla parola precedente/successiva. In seguito per l'estrazione del modello vengono utilizzate le liste `models` e `exception`, in combinazione con la lista `measures`, che contiene alcuni suffissi che indicano che una parola sia alfabetica che numerica che termina con una di essi non deve essere considerata come un modello perché rappresenta una misura (ad esempio, `"mm"` o `"gb"`). Il modello viene estratto usando un'espressione regolare (*regular expression*) concepita per identificare le parole contenenti sia lettere che cifre.

```
# Manage suffixes to make alphanumeric model strings
if len(suffixes) > 0:
        for i in range(1, len(splitted)):
                if (splitted[i] in suffixes) and (splitted[
                   i-1].endswith(splitted[i]) == False):
                        splitted[i - 1] = splitted[i - 1] +
                            splitted[i]

# Manage prefixes to make alphanumeric model strings
if len(prefixes) > 0:
        for i in range(0, len(splitted) - 1):
                if (splitted[i] in prefixes) and (splitted[
                   i+1].startswith(splitted[i]) == False):
                        splitted[i] = splitted[i] +
                            splitted[i + 1]

# List of measure suffixes to be ignored for the model
    extraction
        measures = ["cm", "mm", "nm", "in", "inch",
            "gb", "mb", "mp", "megapixel", "megapixels",
            "mega", "hz", "mah", "cmos", "mps", "cd",
            "ms", "watt", "bit", "res", "resolution",
            "kg"]

# Extract model (no more only among the first 5 words, but
    in the whole string)
```

```
model = 'none'
for s in splitted:
        if ((bool(re.match('^(?=.*[0-9])(?=.*[a-z])', s))
           == True) and (s not in exceptions)) or (s in
           models):
                is_measure = False
                for m in measures:
                        if s.endswith(m):
                                is_measure = True
                if is_measure == False:
                        model = s
                        break
```

Quindi, ulteriori operazioni possono essere eseguite sul modello estratto al fine di risolvere altre cause di dissimilarità. In particolare, può essere eseguita la modifica o la rimozione di alcuni prefissi o suffissi (ad esempio, i suffissi che indicano i colori e che causerebbero falsi negativi), ma anche la gestione delle equivalenze speciali che sono rappresentate dai diversi nomi utilizzati per lo stesso modello a seconda dell'area geografica in cui viene venduto o a seconda di particolari scelte effettuate dal marchio. Queste dissimilarità devono essere conformate attraverso la scelta di un design unico. Inoltre, è anche possibile gestire il problema dei modelli con più generazioni ricorrente per alcune marche: il modello *Sony dscrx100* può essere presente come *Sony dscrx100 2* , *Sony dscrx100 m2* , *Sony dscrx100 iii* , *Sony dscrx100 m3* e devono essere considerati come modelli diversi se appartenenti a due generazioni diverse, mentre ovviamente il passaggio di estrazione del modello precedente potrebbe rilevare solo `dscrx100` senza questo ulteriore dettaglio, generando falsi positivi. Infine, per implementare l'ultimo passaggio della normalizzazione viene utilizzato il dizionario `equivalences`: se il modello trova corrispondenza in tale dizionario viene sostituito con il rispettivo valore normalizzato.

```
if brand == 'sony' and model != 'none':
        if model.startswith('dslr'):
                model = model.replace('dslr', '')
        elif model.startswith('hx'):
                model = model.replace('hx', 'dschx')
        elif model.startswith('ilca'):
                model = model.replace('ilca', 'a')
        elif model.startswith('ilcea'):
            model = model.replace('ilcea', 'a')
```

```
        elif model.startswith('ilce'):
            model = model.replace('ilce', 'a')
        elif model.startswith('ice'):
            model = model.replace('ice', 'a')
        elif model.startswith('mvc'):
            model = model.replace('mvc', '')
        elif model.startswith('p'):
            model = model.replace('p', 'dscp')
        elif model.startswith('qx'):
            model = model.replace('qx', 'dscqx')
        elif model.startswith('rx'):
            model = model.replace('rx', 'dscrx')
        elif model.startswith('slt'):
            model = model.replace('slt', '')
        elif model.startswith('tx'):
            model = model.replace('tx', 'dsctx')
        elif model.startswith('w'):
            model = model.replace('w', 'dscw')
    mods = {'1200tv': '1200tvl', '300k': 'a300',
    '350x': 'a350', '420800tvl': '420tvl',
    '480600tvl': '480tvl', '5n': 'nex5n',
    '600tvllow': '600tvl', '700tv': '700tvl',
    '7r': 'a7r', '7s': 'a7s', 'a350x': 'a350',
    'a37m': 'a37', 'a55vl': 'a55','a58m': 'a58',
    'a65vk': 'a65', 'a65vl': 'a65', 'a65vm': 'a65',
    'a77m2': 'a77 2','a77m2q': 'a77 2', 'a77vm':
    'a77','cd400kitis' : 'cd400', 'dsch7megamovie':
    'dscwx350', 'f707': 'dscf707', 'f828':
    'dscf828', 'h10': 'dsch10', 'h20': 'dsch20',
    'h200': 'dsch200', 'h400': 'dsch400', 'h50':
    'dsch50', 'h90': 'dsch90', 'hdras100vr':
    'hdras100', 'hdrpj240er': 'hdrpj240e',
    'hdrpj240es': 'hdrpj240e', 'hdrpj340ew':
    'hdrpj340e', 'nex3kb': 'nex3', 'nex3nbmbdl':
    'nex3n', 'nex567': 'nex5', 'nex5c': 'nex5',
    'nex5ndslr': 'nex5n', 'nex5rkb': 'nex5r',
    'nex5tls': 'nex5t', 'nex6lb2bdl': 'nex6',
    'nex6lb': 'nex6', 's2100': 'dscs2100',
    's50': 'dscs50', 's650': 'dscs650', 's70':
    'dscs70', 's85': 'dscs85', 'slta55': 'a55',
    'stla99v': 'a99', 't300': 'dsct300', 't90':
    'dsct90', 't99': 'dsct99'}
    if model in mods.keys():
```

```
                model = mods[model]
        if model.endswith('b'):
                model = model.replace('b', '')
        elif model.endswith('k'):
                model = model.replace('k', '')
        elif model.endswith('l'):
                model = model.replace('l', '')
        elif model.endswith('v'):
                model = model.replace('v', '')
        elif model.endswith('y'):
                model = model.replace('y', '')
        if model == 'a77':
        if ' ii ' in camera['page_title'] or ' 2 ' in
           camera['page_title']:
                model = model + ' ' + '2'
        if model.startswith('dscrx100'):
        if ' ii ' in camera['page_title'] or model in ['
           dscrx1002', 'dscrx100m2']:
                model = 'dscrx100' + ' ' + '2'
        elif ' iii ' in camera['page_title'] or model in ['
           dscrx100iii', 'dscrx100m3']:
                model = 'dscrx100' + ' ' + '3'

# Resolve equivalences
if model in equivalences.keys():
        model = equivalences[model]
```

È importante notare che tutti i modelli e le caratteristiche tipiche di ogni marca (gestione dei prefissi e dei suffissi, eccezioni, linee diverse per lo stesso prodotto, ecc.) sono stati gestiti manualmente attraverso lo studio dei dati. Ad esempio, per quanto riguarda la gestione dei suffissi dei modelli, non sono stati rimossi quelli che si riferivano a diverse versioni dello stesso modello o a diverse caratteristiche tecniche (che sono state considerate come elementi atti a definire in modo univoco un determinato modello). Come delucidazione è fornito l'esempio del monitor Dell 1907FP, disponibile in tre diverse versioni (1907FPc, 1907FPt e 1907FPv), che si differenziano per la presenza di pannelli di `Samsung`, AUO, o LG. Ovviamente, i differenti pannelli implementati producono delle differenze tecniche nelle tre versioni del modello 1907FP. Nella *Figura 3.1* sono riportate le specifiche dei tre diversi pannelli. Al contrario, i suffissi che non si riferiscono a caratteristiche tecniche sono stati rimossi (ad esempio il suffisso `b` per indicare il colore `black`).

Samsung
LTM190EX

Features

Item	Description
Size	19.0"
Resolution	SXGA
Number of Pixels	1,280 x 1,024
Active Area(mm)	376.3 x 301.1
Pixel Pitch(mm)	0.294
Mode	B-TN III
Number of Colors	16.7M
Contrast Ratio(typ.)	1,000:1
Brightness(cd/m^2)	300
Response Time(ms at 25°C)	5(on/off)
Viewing Angle(U/D/L/R)	80/80/80/80
Interface	LVDS (2Ch)
Outline Dimension(mm)	396.0 x 324.0 x 16.5
Weight(g)	2,500
Production	1Q.'06
Color Gamut(%)	72

AUO

Size	19"
Model	M190EN04 V5
Resolution (pixel)	SXGA (1280 x 1024)
Aspect Ratio	5 : 4
Active Area (mm)	376.3 x 301.1
Pixel Pitch (mm)	0.294
Mode	TN
Number of Colors	16.2M
Color Saturation (%)	72
View Angle (H/V)	150 / 135
Brightness (cd/m²)	270
Contrast Ratio	500 : 1
Response Time (ms) (at 25°C)	8
Power Consumption (W)	28.0
Interface	2ch LVDS
Supply Voltage (V)	5
Backlight	4 CCFL
Outline Dimensions (mm)	396.0 x 324.0 x 17.5
Weight (g)	2500
Production	Now
Remark	Narrow Bezel

LG/Philips
LM190E03

- Features

19" TFT LCD Panel for Monitor

Model Name	LM190E03
Active Area [mm]	376.3 x 301.1
Outline Dimension [mm]	404.2 x 330
Thickness [mm]	20
Resolution	1,280 x RGB x 1,024
Aspect Ratio	5:4
Pixel Pitch [mm]	0.294(86.4)
Number of Colors	16.2M(6bit+FRC)
Luminance [cd/㎡]	250/400
Color Saturation (%)	72%
Weight [g]	2,500
Contrast Ratio	500:1
Interface	LVDS
Viewing Angle [°,U/D/L/R]	140/140
Color Temperature [K]	
Response Time [ms]	12
MP Schedule	

Figura 3.1: Specifiche dei pannelli Dell 1970FP

Una volta estratti e normalizzati i valori di marchio e modello dalle specifiche dove è stato possibile rilevarli, si procede con la seconda perte dell'algoritmo, ovvero la fase di matching. L'attributo `page_title` viene ridefinito come l'unione delle parole presenti in `splitted`, al fine di includere le modifiche applicate, e viene aggiunto un nuovo attributo: `brand_n_model`, che contiene la concatenazione del marchio e del modello estratti. Ovviamente, questo accade solo nel caso in cui siano stati rilevati entrambi. In tal caso, la tupla (espressa tramite il dizionario `monitor`) viene aggiunta all'elenco `solved_specs` in caso contrario, la tupla viene aggiunta all'elenco `unsolved_specs`.

```
monitor['page_title'] = ' '.join(splitted)

# Put the current specification in the right group (solved
    or unsolved)
if (brand != 'none') and (model != 'none'):
        monitor['brand_n_model'] = brand + ' ' + model
        solved_specs.append(monitor)
else:
        unsolved_specs.append(monitor)
```

Per determinare le coppie di tuple corrispondenti il primo passaggio è applicare l'indice inverso (inverted index) a `solved_specs` per generare i blocchi, raggruppando gli elementi in base alla perfetta corrispondenza dei rispettivi attributi `brand_n_model`. In seguito viene calcolato il prodotto cartesiano tra gli elementi dello stesso blocco, escludendo il caso di identità (prodotto di una tupla con se stesso) e gestendo la riflessività. Ovvero, aggiungendo le coppie generate a un set solo dopo aver ordinato i due elementi che lo compongono, in modo che le coppie riflessive siano viste come duplicati e non vengano inserite nuovamente nel set. Ovviamente, poiché la loro generazione si basa sul valore esatto dell'attributo `brand_n_model`, i blocchi sono disgiunti. L'insieme di coppie viene successivamente inserito in un `DataFrame Pandas` (tipo utilizzato per emulare tabelle) costituito da due colonne (`left_spec_id`, `right_spec_id`), in cui ogni riga definisce una coppia di specifiche corrispondenti. Tale dataframe sarà poi salvato in un file di tipo `CSV`.

```
# Get matches from solved specifications
clusters = dict()
```

```
for s in solved_specs:
        if s['brand_n_model'] in clusters.keys():
                clusters[s['brand_n_model']].append(s['id'
                    ])
        else:
                clusters.update({s['brand_n_model']:[s['id'
                    ]]})

couples = set()
for c in clusters.keys():
        if len(clusters[c]) > 1:
                for i in clusters[c]:
                        for j in clusters[c]:
                                if i < j:
                                        couples.add((i, j))
                                if i > j:
                                        couples.add((j, i))

couples = list(couples)
couples = pd.DataFrame(couples, columns = ['left_spec_id',
    'right_spec_id'])
```

Per quanto riguarda la lista delle specifiche ove non sono stati estratti marchio o modello (`unsolved_specs`), la metodologia di rilevazione delle corrispondenze si basa sull'attributo `page_title` (a cui sono state applicate le modifiche precedentemente descritte). Le specifiche visualizzate in questo elenco vengono considerate come corrispondenze solo se il contenuto dell'attributo `page_title` è esattamente identico: ancora una volta, viene utilizzato l'inverted index per generare i blocchi in base alla corrispondenza di tale attributo. Le coppie corrispondenti, trovate seguendo il metodo descritto nel caso precedente, vengono inserite in un `DataFrame Pandas`.

```
# Find identical strings in unsolved specifications
clusters = dict()

for u in unsolved_specs:
        if u['page_title'] in clusters.keys():
                clusters[u['page_title']].append(u['id'])
        else:
                clusters.update({u['page_title'] : [u['id'
                    ]]})
```

```
identities = set()
for c in clusters.keys():
        if len(clusters[c]) > 1:
                for i in clusters[c]:
                        for j in clusters[c]:
                                if i < j:
                                        identities.add((i,
                                            j))
                                if i > j:
                                        identities.add((j,
                                            i))

identities = list(identities)
identities = pd.DataFrame(identities, columns =
    ['left_spec_id', 'right_spec_id'])
```

In conclusione, i due set di dati vengono concatenati e il dataframe risultante è salvato in un file in formato `CSV`: tale file sarà usato per valutare l'F-measure.

```
couples = pd.concat([couples, identities])

couples.to_csv('matches_from_solved.csv', index=False)
```

3.4 Risultati

Una volta ricevuto il ground truth dagli organizzatori del contest di DI2KG che prende in considerazione 2272 specifiche, abbiamo utilizzato tali informazioni per valutare precision e recall del nostro algoritmo. Considerando il dataset dopo averlo sottoposto ad un primo studio generico, con l'introduzione di modifiche non mirate, i risultati ottenuti sono stati 0.88 per la precision e 0.89 per la recall. Nonostante la genericità dello studio iniziale si tratta comunque di un buon risultato, in particolare se commisurato al lavoro umano (criticità predominante del nostro algoritmo) necessario al suo raggiungimento. Ricordo che l'approccio su cui si basa questo algoritmo è scaturito dal fatto che le tecniche di machine learning e deep learning non producevano risultati soddisfacenti su questa tipologia di problemi, in particolare a causa di un eccesso di falsi positivi.

Avendo dunque a disposizione il ground truth abbiamo potuto procedere all'inserimento di correzioni mirate al fine di ridurre il numero di falsi positivi e negativi, migliorando i valori di precision e recall. Il primo passaggio è stato la risoluzione mirata delle eccezioni che producevano una non corretta risoluzione di alcuni record. Ad esempio per il record `www.vology.com//2203`, con valore di `page_title` "hp `d7q14a8#aba` z display `z22i` monitor 21.5 vology" inizialmente si rilevava come modello la stringa `d7q14a8#aba`, che però rappresenta il codice dell'articolo, inserendo tale stringa nelle eccezioni veniva poi correttamente rilevata la stringa `z22i` come modello, eliminando delle coppie rilevate come falsi negativi e migliorando la recall. Il secondo passaggio consiste nella risoluzione degli alias, la rimozione di suffissi e la correzione di errori di battitura, fattori che determinano un buon numero di falsi negativi, inficiando la recall. Portiamo come esempio il modello `f75aty133h` della `Asus` che presenta il suffisso `ty133h`, non atto a determinare il modello bensì le singole specifiche del prodotto, considerate non discriminanti dal punto di vista degli organizzatori. Un esempio di alias è invece rappresentato dalla definizione dell'equivalenza tra `compaq` e `hp` (in quanto il brand Compaq è stato acquisito dal gruppo Hewlett-Packard) al fine della risoluzione dei record. Dopo questi primi passaggi il valore della recall è migliorato fino a raggiungere lo 0.965. Un fattore limitante per un ulteriore miglioramento era la presenza di record che non venivano risolti in quanto non presentavano in modo esplicito informazioni su marca o modello. Il primo approcco per la risoluzione di tali record è stato basato sull'inserimento in coda all'attributo `page_title` di eventuali campi conteneti informazioni rilevanti su marchio e modello (`brand`, `manufacturer`, `model`), che assumono una rilevanza solo nel caso in cui queste informazioni non siano già state reperite nella stringa `page_title`. Questo ha richiesto di apportare una modifica all'inizio del

nostro codice, ovvero durante la lettura di ogni singola specifica, per poter attuare anche su questi campi le stesse procedure di normalizzazione già descritte in precedenza.

```
#Concatenate eventual attributes 'brand', 'manufacturer',
    and 'model' to page_title
if 'brand' in spec.keys() and type(spec['brand']) == str:
        spec['<page title>'] = spec['<page title>'] + spec[
            'brand']
if 'manufacturer' in spec.keys() and type(spec['
    manufacturer']) == str:
     spec['<page title>'] = spec['<page title>'] + spec['
         manufacturer']
if 'model' in spec.keys() and type(spec['model']) == str:
        spec['<page title>'] = spec['<page title>'] + spec[
            'model']
```

Questo approccio ha consentito di risolvere solo una parte dei record che non venivano risolti a causa di informazioni non esaurienti per quanto concerne marchio e modello. È stato perciò necessario uno studio più specifico su quelle tuple che non presentavano i campi di nostro specifico interesse né in `page_title` né in attributi distinti. Avendo notato che alcuni modelli sono caratteristici di precise marche, è possibile da questa singola informazione costruire l'attributo `brand_n_model`, forzando l'assegnamento del marchio. Un esempio è riportato di seguito.

```
# Put the current specification in the right group (solved
    or unsolved)
if camera['id'] in gt_records:
        if (brand != 'none') and (model != 'none'):
                camera['brand_n_model'] = brand + ' ' +
                    model
                solved_specs.append(camera)
        elif (brand == 'none') and (model != 'none'):
                if  (model == 'v176l'):
                        brand = 'acer'
                elif  (model == '1715l'):
                        brand = 'elo'
                elif  (model == 'p17a'):
```

```
                brand = 'hp'
            elif (model == 'bl702a'):
                brand = 'benq'
            elif (model == 'pa242w'):
                brand = 'nec'
        if (brand != 'none'):
            camera['brand_n_model'] = brand + '␣' +
                model
            solved_specs.append(camera)
    else:
    unsolved_specs.append(camera)
```

Al termine di tale procedimento la recall è arrivata al valore di 0,972, risultato soddisfacente considerata la natura dei dati trattati. Difatti, non è stato possibile incrementarlo ulteriormente, seguendo il nostro metodo, dal momento che erano stati inclusi nel ground truth alcuni record in cui non era possibile determinare in modo univoco l'entità corrispondente.

Per quanto riguarda invece la precision, il suo valore è cresciuto molto velocemente a seguito della correzione di imprecisioni ed errori di battitura, fino a giungere al valore di 0,999. Analizzando le rimanenti coppie di record individuate come falsi positivi, abbiamo rilevato la presenza di due significativi errori nel ground truth: in particolare il record `www.ebay.com//9783`, il cui attributo `page_title` contiene la stringa "samsung samsung s22c650d 21 5 inch screen lcd monitor 887276017808 ebay" per cui è evidente la rispettiva coppia marca-modello (`samsung s22c650d`), era invece attribuito all'entità cui facevano capo record aventi `brand_n_model: nec v801` (ad esempio `www.ohc24.ch//363`). Analoga situazione si verifica per il record `www.ebay.com//22284` con attributo `page_title` "lenovo thinkvision 9417hc2 17 lcd monitor vga db15 1280 x 1024 damaged 882861738410 ebay" e attributo `brand_n_model: lenovo 9417hc2`, che veniva accorpato a record riferiti all'entità identificata come `philips 278c4qhsn`. A seguito della correzione di tali errori di attribuzione la precision è arrivata al valore del 100%. Possiamo ora calcolare il valore di F-score che risulta essere pari a 0,985.

Capitolo 4

Conclusione

Nell'ambito della presente tesi abbiamo descritto il problema dell'*entity resolution*, focalizzandoci in particolare su un contesto concreto presentato nell'ambito del contest di programmazione legato al workshop DI2KG. In primo luogo abbiamo trattato le tecniche che rappresentano lo stato dell'arte in questo ambito, che tuttavia presentano ancora limiti rilevanti in contesti come quello trattato. In seguito abbiamo esposto un metodo di risoluzione alternativo che consta di un algoritmo basato sulle espressioni regolari.

Tale algoritmo affronta l'*entity matching* sul dataset fornitoci nell'ambito del contest precedentemente citato, composto da record riguardanti monitor e loro accessori. Per risolvere tali record nelle rispettive entità cui si riferiscono abbiamo constatato che le informazioni rilevanti per tale risoluzione fossero la marca e il modello di ogni prodotto, entrambi solitamente contenuti nell'attributo principale `page title`, comune a tutti i record. Tali informazioni venivano estratte grazie ad una lista di marche comuni per il brand e un'espressione regolare per il modello (che nel più dei casi è una stringa sia numerica che alfabetica). Queste operazioni sono state accompagnate da tecniche di normalizzazione e trattamento delle eccezioni e criticità riscontrate durante lo studio dei dati, al fine di migliorare l'accuratezza dell'estrazione e le prestazioni dell'algoritmo.

I risultati di questo algoritmo, valutati a partire dal ground truth fornito dagli stessi organizzatori del contest, sono stati molto soddisfacenti e hanno confermato la validità dell'approccio che esula dall'applicazione machine learning e deep learning per questa tipologia di problemi di *entity resolution*. A fronte di un lavoro umano di studio dei dati mirato alla risoluzione delle singole criticità relativamente limitato in termini temporali e focalizzato sui dati presenti nel ground truth, la precision e la recall calcolate sono state rispettivamente 100% l'una e 97% l'altra (valore fortemente influenzato dalla natura 'sporca' dei dati). È importante notare che lo studio mirato dei dati

del ground truth non è stato determinante per il raggiungimento di risultati soddisfacenti, in quanto già uno studio più generico e superficiale del dataset in esame aveva prodotto valori di precision e recall molto vicini al 90%. Questo risultato evidenzia la validità dell'algoritmo presentato e predispone le basi per una possibile estensione futura a problemi con esigenze computazionali superiori e una maggiore diversificazione dei dati.

Bibliografia

[1] Luca Gagliardelli et al. «SparkER: Scaling Entity Resolution in Spark». In: *EDBT 2019, Lisbon, Portugal, March 26-29, 2019*. 2019, pp. 602–605.

[2] Giovanni Simonini et al. «Scaling entity resolution: A loosely schema-aware approach». In: *Inf. Syst.* 83 (2019), pp. 145–165.

[3] Giovanni Simonini et al. «Schema-Agnostic Progressive Entity Resolution». In: *ICDE 2018*. 2018.

[4] Giovanni Simonini, Sonia Bergamaschi e H. V. Jagadish. «BLAST: a Loosely Schema-aware Meta-blocking Approach for Entity Resolution». In: *Proc. VLDB Endow.* 9.12 (2016), pp. 1173–1184.

[5] H. Altwaijry, S. Mehrotra e D. V. Kalashnikov. «QuERy: A Framework for Integrating Entity Resolution with Query Processing». In: *Proceedings of the VLDB Endowment* 9.3 (2015), pp. 120–131.

[6] H. Altwaijry, D. V. Kalashnikov e S. Mehrotra. «QDA: A Query-Driven Approach to Entity Resolution». In: *IEEE Transactions on Knowledge and Data Engineering* 29.2 (2017), pp. 402–417.

[7] A. Pietrangelo et al. «Towards Progressive Search-Driven Entity Resolution». In: *26th Italian Symposium on Advanced Database Systems, SEBD 2018* (2018).

[8] S. Mudgal et al. «Deep Learning for Entity Matching: A Design Space Exploration». In: *Proceedings of the 2018 International Conference on Management of Data* (2018), pp. 19–34.

[9] G. Amici, S. Bergamaschi e F. Naumann. «A Query-Driven Approach to Entity Resolution Based on Data Ordering».

[10] L. Zecchini, G. Simonini e S. Bergamaschi. «Riconciliazione progressiva di entità guidata dalle interrogazioni».

[11] *ACM SIGMOD/PODS 2020*. URL: `https://sigmod2020.org`.

[12] *ACM SIGMOD 2020 Programming Contest*. URL: http://www.inf.uniroma3.it/db/sigmod2020contest.

[13] *Magellan*. URL: https://sites.google.com/site/anhaidgroup/projects/magellan.

[14] *DeepMatcher*. URL: https://github.com/anhaidgroup/deepmatcher.

[15] *DELL technologies*. URL: https://www.dell.com.

[16] *Wikipedia, the free encyclopedia*. URL: https://en.wikipedia.org.

**

Appendice A

Ambiente

In questa appendice, oltre all'ambiente utilizzato per la produzione del codice, ad esempio compilatore, linguaggio di programmazione e le API, sono illustrati gli strumenti utilizzati per l'esecuzione dei test, ad esempio l'hardware.

A.1 Specifiche

Le caratteristiche computazionali della macchina utilizzata per l'esecuzione di test e controlli sono le seguenti:

PROCESSORE: Intel Core i5-7200U 2.50Ghz
RAM: 8Gb
SISTEMA OPERATIVO: Windows 10 Home 64bit

A.2 Python

Figura A.1: Python logo

Python è un linguaggio di programmazione interpretato, orientato agli oggetti, di alto livello, adatto tra gli altri usi a sviluppare applicazioni distribuite, scripting, computazione numerica e system testing. Le sue strutture di

alto livello insieme al dynamic typing e dynamic binding, lo rendono adatto per lo sviluppo rapido (un programma in Python rispetto ad un programma Java è tipicamente da 3 a cinque volte più corto).

Ideato da Guido van Rossum all'inizio degli anni novanta, partendo dall'ABC, un linguaggio didattico monolitico che non prese piede negli anni '80.

Se paragonato ai linguaggi compilati *statically typed*, come ad esempio il C, la velocità di esecuzione non è uno dei punti di forza di Python, che si pone però come obbiettivi dinamicità, semplicità, flessibilità e permette così di ridurre il tempo di sviluppo.

A.3 Pandas

Figura A.2: Pandas logo

Pandas è uno strumento di analisi e manipolazione dei dati open source veloce, potente, flessibile e facile da usare, basato sul linguaggio di programmazione Python. Questa libreria fornisce gli strumenti per l'analisi dei dati, infatti viene fornita con diverse strutture dati che possono essere utilizzate per diverse attività di manipolazione dei dati. Pandas è una libreria molto usata per il recupero e la preparazione dei dati per l'uso futuro in altre librerie Machine Learning come Scikit-learn o Tensorflow. Permette inoltre di recuperare facilmente i dati da diverse fonti: database SQL, testo, CSV, Excel, file JSON e molti altri formati meno popolari. Una volta che i dati sono in memoria, ci sono dozzine di operazioni diverse per analizzare, trasformare, recuperare i valori mancanti, pulire il set di dati, nonché operazioni tipo SQL e un set di funzioni statistiche per eseguire anche una semplice analisi.