REPORT SECONDO PROGETTO BIG DATA

Fairness in Exposure of Rankings

THE

Daniele Caldarini
452241

Michele Tedesco
470338

PRESENTAZIONE

Nel seguente report andremo a descrivere il lavoro svolto per il secondo progetto del corso di Big Data.
Lo scopo del progetto è stato quello di implementare e testare le tecniche descritte nel paper *Fairness of Exposure in Rankings* di Ashudeep Singh e Thorsten Joachims.

Nel seguito si esporranno i concetti principali descritti nel paper per introdurre il lavoro, successivamente sarà descritto in vari passi il lavoro di implementazione e test svolto.
In particolare segue una descrizione dei dataset utilizzati, una descrizione dell'architettura dell'applicazione sviluppata, una descrizione più dettagliata dei moduli principali e infine una sezione con la presentazione e l'analisi degli esperimenti effettuati.

Il codice del lavoro svolto, così come file e istruzioni per la creazione e il provisioning dell'ambiente di esecuzione sono presenti al repository di Github al seguente link:
https://github.com/Meyke/Progetto2BigDataFlask

INTRODUZIONE

La ricerca e la presentazione di risultati sotto forma di ranking è presente in moltissime applicazioni.

Nell'esposizione di un ranking si hanno delle responsabilità non solo verso gli utenti ma anche verso gli item presenti nel ranking.

Le due responsabilità sono solitamente in conflitto tra loro: restituire all'utente risultati che siano utili in termini di rilevanza dell'utente, spesso va a discapito dell'equità nella presentazione dei risultati.

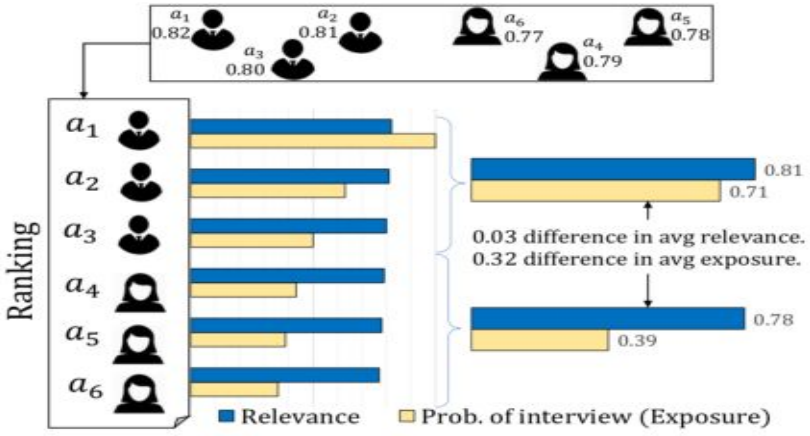


Fig.1: Job seeker example to illustrate how small a difference in rilevance can lead to a large difference in exposure for a group of females

Nell'esempio raffigurato in *fig.1* si vogliono raccomandare candidati a un datore di lavoro per svolgere una mansione. Tale datore di lavoro utilizza un algoritmo che ritorna i candidati più idonei nell'eseguire una certa mansione. L'algoritmo restituisce i risultati esclusivamente per ordine di rilevanza, senza badare a un'ipotetica *fairness*. In particolare, al datore vengono restituiti 6 possibili candidati con associato un valore di rilevanza. È possibile notare come anche un piccola differenza in termini di rilevanza sfoci in una non trascurabile differenza in termini di *exposure* dei risultati. Utilizzando come metrica, per esempio, la Discounted Cumulative Gain, i candidati donna risultano essere meno esposti nel ranking minore del 30% rispetto a quella degli uomini. Questo lo si può anche notare anche guardando le semplici posizioni in classifica delle candidate donne rispetto ai candidati uomini.

Un ranking che cerca di massimizzare il grado di utilità per gli utenti tende a mantenere l'ordine di presentazione dei candidati in termini di rilevanza, ma volendo rispettare eventuali vincoli di fairness e garantire quindi un equità maggiore (nel caso in esame dare più risalto ai candidati donna), il problema diventa quello di massimizzare il grado di utilità, condizionato però al rispetto di alcuni vincoli con l'obiettivo di garantire una distribuzione più *fair* dei risultati.

Il concetto di fairness, o equità, nei ranking è un concetto molto delicato da affrontare. Non esiste una definizione universale di fairness in quest'ambito, ma varia a seconda dell'applicazione, del contesto e del grado di applicazione.
In particolare nel paper vengono proposti tre tipi di vincoli di fairness, tutti con l'obiettivo di garantire equità tra gruppi di item. Non viene presa in considerazione invece la fairness individuale, poiché è difficile da definire a causa della mancanza di accordo su possibili metriche di similarità individuale per uno specifico task.

Nel caso del job seeker descritto precedentemente, si possono facilmente individuare due gruppi: uomini e donne. Ma nulla vieta di poter raggruppare individui sulla base di altre features, quali etnia o età, ottenendo magari anche più di due gruppi.
Lo stesso ragionamento può essere esteso anche a gruppi di item che non sono persone.

Nell'ambito del progetto viene preso in considerazione un dataset di news individuando e dividendo gli item per gruppi sulla base della testata giornalistica che ha pubblicato la news. I vincoli di fairness in questo dominio dovranno restituire notizie rilevanti per una certa query ma che siano allo stesso tempo ben distribuite rispetto alle varie testate.
Come detto in precedenza, i vincoli di fairness e il trade-off tra utilità ed equità sono concetti che variano a seconda del contesto. Il progetto che andremo a descrivere si pone come obiettivo proprio quello di implementare diversi tipi di vincoli e poi utilizzarli al fine di massimizzare l'utilità, del ranking restituito, con riferimento ai vincoli stessi.

Consideriamo un set di documenti e una query q. Si indica con $U(r|q)$ l'utilità di un ranking data la query q. Quello che andremo a risolvere è un problema di *massimizzazione*,

estendendo il caso senza fairness al caso che prende in considerazione invece dei precisi vincoli di fairness. In breve, possiamo dividere i passi da svolgere in:

1. estendere il concetto di ranking, che di per sé è un concetto combinatorio, al concetto di ranking probabilistico;
2. riformulare il problema di ottimizzazione (massimizzazione) come un problema di programmazione lineare, aggiungendo a questo dei vincoli di fairness descritti con un opportuno linguaggio;
3. ricostruire il ranking a partire dalla soluzione del problema di programmazione lineare;

DATASET

In questa sezione andiamo a descrivere i dataset utilizzati per il nostro progetto.

Il primo dataset utilizzato è un dataset di news preso da Kaggle. Il dataset è presente al seguente link https://www.kaggle.com/snapcrack/all-the-news.

Le news sono principalmente comprese tra gli anni 2016 e luglio 2017, anche se c'è un numero non trascurabile di articoli a partire dal 2015.

In particolare il dataset consiste di 143 mila articoli provenienti da 15 diverse pubblicazioni.

Il dataset consiste di 3 file csv e gli articoli sono così divisi:

- **articles1.csv** - 50,000 news articles (Articles 1-50,000)
- **articles2.csv** - 49,999 news articles (Articles 50,001-100,00)
- **articles3.csv** - Articles 100,001+

Per quanto riguarda la distribuzione rispetto alle diverse pubblicazioni presenti possiamo osservare il seguente grafico:

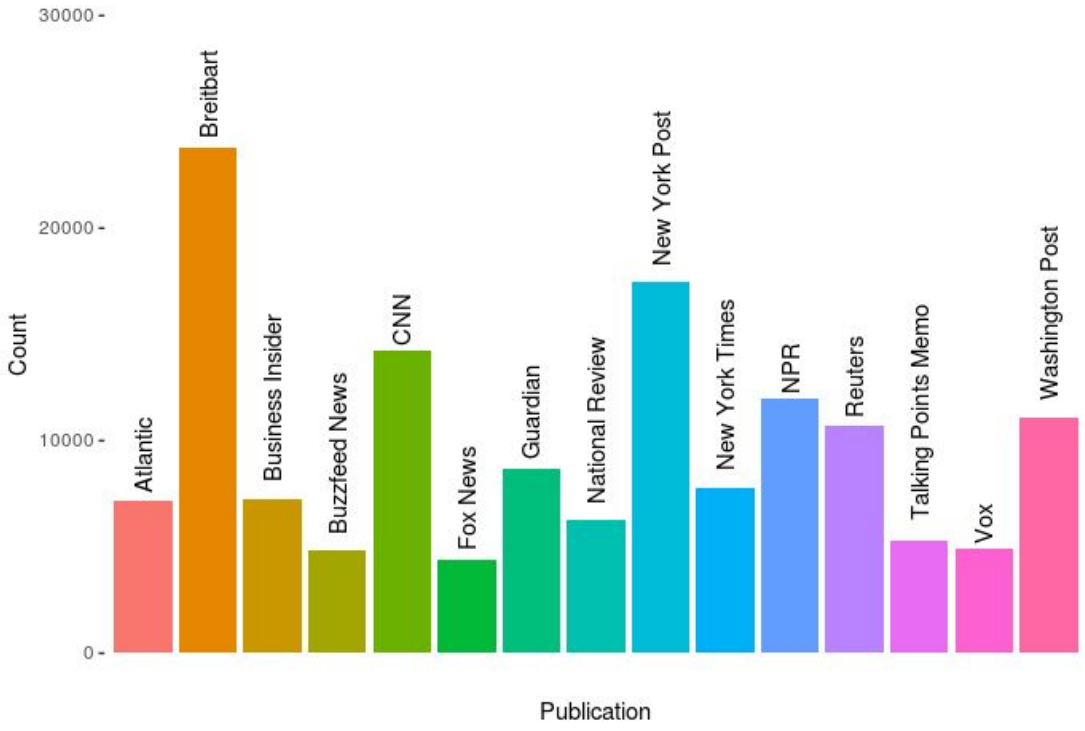


Fig.2 Distribuzione dataset news

La distribuzione per alcune testate è sbilanciata, così come non è dato sapere con certezza se i diversi argomenti siano trattati in modo uniforme.
Per i nostri scopi tutto ciò non rappresenta un problema, anzi può essere utile per avere risultati poco equi, quindi con uno squilibrio di notizie provenienti da una data testata, per poi applicare successivamente i vincoli di fairness al fine di produrre un risultato più equo.
Ogni riga dei file csv rappresenta una news e per ogni news sono riportate le seguenti informazioni:

- id
- title: titolo dell'articolo
- publication: nome della pubblicazione/testata
- author: nome dell'autore dell'articolo
- date: data di pubblicazione dell'articolo
- url: url dell'articolo(non sempre presente)
- content: contenuto dell'articolo

ARCHITETTURA

L'architettura della nostra applicazione è basata sull'utilizzo di contenitori Docker e ambienti virtuali in Python. In particolare possiamo individuare tre blocchi principali facendo uso delle seguenti tecnologie: Spark, Elasticsearch e Flask.
Di seguito un'illustrazione dell'architettura.

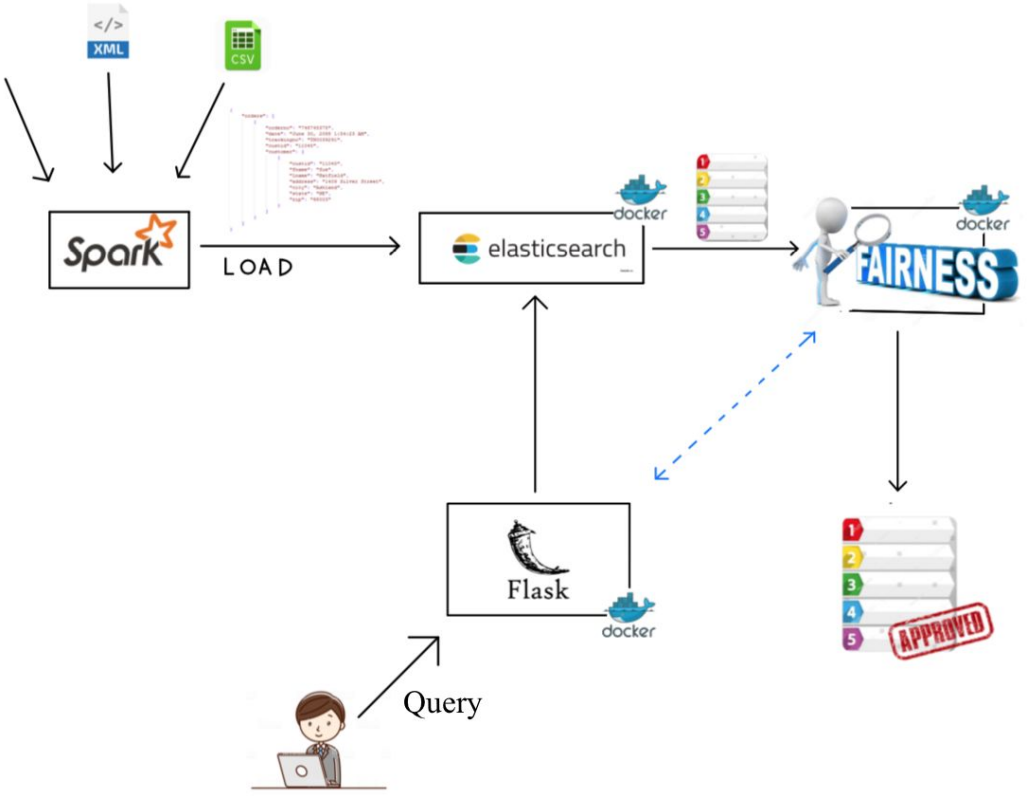


Fig.3: Architettura applicazione fairness

Apache Spark viene utilizzato per caricare in modo efficiente e distribuito i diversi dataset su un indice di Elasticsearch. In particolare è stato utilizzato SparkSQL con pyspark come strumento. A partire dal csv del dataset viene creato un dataframe, che viene successivamente filtrato e poi scritto su Elasticsearch grazie a opportune librerie che permettono a Spark di poter comunicare in modo efficiente con Elasticsearch.

È possibile creare e lanciare un contenitore con Spark installato e connetterlo tramite una network Docker al contenitore di Elasticsearch (che ascolta sulla porta 9200), che rimane poi in ascolto per le query che arrivano dall'app sviluppata con Flask e che è accessibile sulla porta 5000. I vari contenitori docker comunicano tra loro tramite chiamate HTTP REST.
L'applicazione in Flask viene eseguita all'interno di un ambiente virtuale di Python.
Data una query, Flask ci permette di inviarla in modo semplice e diretto attraverso ai contenitori in cui è in esecuzione Elasticsearch, che restituisce un ranking dei risultati più rilevanti, ordinati rispetto ad uno score che altro non è che un numero di tipo float positivo. Tale score viene calcolato tramite metriche quali TF-IDF o basate su cosine similarity. Per i nostri scopi abbiamo utilizzato il modello e lo score di default di Elasticsearch (https://www.elastic.co/guide/en/elasticsearch/guide/2.x/practical-scoring-function.html**).**
Il ranking restituito e i corrispondenti valori di rilevanza vengono quindi utilizzati e inviati ad un modulo Python, che per semplicità chiamiamo modulo *fairness*) che si occuperà di applicare le varie tecniche descritte nell'introduzione al fine di risolvere il problema di ottimizzazione e restituire un nuovo ranking che prenda in considerazione anche i vincoli di fairness scelti.
Il codice di questo modulo, che rappresenta la vera e propria implementazione del framework descritto nel paper, è descritto in modo più dettagliato nell'apposita sezione più avanti.

STRATEGIA E IMPLEMENTAZIONE MODULO DI FAIRNESS

L'implementazione del modulo di *Fairness* è stata scritta in Python e si trova nei seguenti file nel repository:

- fairnessModule.py
- findProbMatrix.py

Si è proceduto implementando i seguenti passi:

1. Individuazione del vettore di utilità *u* e del vettore di discount *v*

Il vettore *u* è che il vettore contenente i valori degli score ritornato da Elasticsearch, su cui viene effettuata una normalizzazione per ridurre l'intervallo dei valori tra 0 e 1 ed avere a disposizione un ranking probabilistico. Il vettore V è invece bla bla bla.

Il file fairnessModule.py contiene un metodo *fairnessMethod(hits, type)* che viene invocato dall'applicazione quando viene richiesto di fare una ricerca rispetto ad una certa query che prenda in considerazione un certo insieme di vincoli di fairness.

Il metodo prende come parametri gli item, ordinati per rilevanza, che Elasticsearch restituisce per la data query(risultati senza fairness).

Ogni elemento della collezione hits è una news con le relative informazioni, con l'aggiunta dello score che Elasticsearch gli assegna. Il parametro type è un identificativo del tipo di vincoli di fairness che verranno applicati durante la risoluzione del problema di massimizzazione.

Il metodo *fairnessMethod* invoca il metodo *findProbMatrix* che inizialmente calcola il vettore *u*, il vettore *v* e infine costruisce una mappa che per ogni pubblicazione (nel nostro caso i gruppi sono proprio le diverse pubblicazioni) calcola il numero di occorrenze di articoli di quella pubblicazione nel ranking, la somma dei valori di utilità e le posizioni nel ranking (qui si fa riferimento ancora al ranking unfair). Questa mappa sarà utilizzata al passo successivo per la costruzione dei vincoli di fairness.

2. Porre il problema in termini di problema di ottimizzazione su vincoli di fairness

Il problema in particolare va posto nella seguente forma:

$$\begin{aligned}
\mathbf{P} = \operatorname{argmax}_{\mathbf{P}} \ & \mathbf{u}^T \mathbf{P} \mathbf{v} && \text{(expected utility)} \\
\text{s.t.} \ & \mathbb{1}^T \mathbf{P} = \mathbb{1}^T && \text{(sum of probabilities for each position)} \\
& \mathbf{P}\mathbb{1} = \mathbb{1} && \text{(sum of probabilities for each document)} \\
& 0 \leq \mathbf{P}_{i,j} \leq 1 && \text{(valid probability)} \\
& \mathbf{P} \text{ is fair} && \text{(fairness constraints)}
\end{aligned}$$

I primi tre tipi di vincoli sono vincoli che fanno si che la matrice di probabilità risultante dalla risoluzione del problema sia una matrice doppiamente stocastica.
I vincoli del quarto tipo sono i vincoli di fairness.
In particolare nel paper si parla di tre tipi di vincoli che possono essere formulati attraverso il framework:

1. **Demographic Parity Constraints**: è il modo più semplice di condizionare la fairness tra gruppi. Quello che si fa è imporre dei vincoli in modo da far si che l'esposizione media dei vari gruppi sia uguale. Questo tipo di vincoli a volte può portare ad una grande perdita di utilità, poiché non considera appunto il grado di utilità dell'item, ma solo il numero di elementi per ogni gruppo cercando appunto di equilibrarlo.
2. **Disparate Treatment Constraints**: questo tipo di vincoli prendono in considerazione anche la rilevanza dei diversi item, cercando di migliorare la fairness rispetto ai vari gruppi in modo proporzionale all'utilità media di quest'ultimi. Vincoli di questo tipo per esempio possono essere molto utili per avere un risultato più fair nel caso del job seeker precedentemente esposto. L'utilizzo di quella che è la probabilità media permette di cercare di limitare quella che in quest'esempio è la grande differenza in termini di esposizione tra i due gruppi, nonostante la piccola differenza in termini di rilevanza.
3. **Disparate Impact Constraints**: questo tipo di vincoli guardano un po più in avanti riflettendo in un certo senso quello che sarà l'impatto di un certo ranking. Di fatto questi vincoli cercano di forzare quello che è il tasso di click previsto per gli item di ogni gruppo, ad essere proporzionale alla propria utilità media.

L'implementazione per quanto riguarda la formulazione e la successiva risoluzione del problema di programmazione lineare sono presenti nella seconda parte del metodo *findProbMatrix.*

Per svolgere queste attività sono state utilizzate le seguenti librerie di Python:

- cvxpy: risoluzione di problemi di ottimizzazione
- numpy: calcolo matriciale
- https://github.com/btaba/sinkhorn_knopp: tecnica utilizzata per adattare i valori della matrice al rispetto di determinati vincoli

Attraverso l'utilizzo della libreria *cvxpy* viene formulato il problema di massimizzazione e tutti i vincoli del problema. Per la formulazione dei vincoli di fairness(a seconda del tipo di vincoli specificato) vengono utilizzati i valori presenti nella mappa costruita al passo precedente.

Successivamente attraverso il metodo solve della libreria viene risolto il problema che produce la matrice dei valori P, che viene adattata attraverso la tecnica di Sinkhorn Knopp al fine di avere sempre una matrice stocastica in output.

3. Calcolo della decomposizione di Birkhoff-von Neumann

La soluzione P del problema di programmazione lineare è una matrice contenente le probabilità di ogni documento a ogni posizione. Per utilizzare questa soluzione in un sistema di ranking, bisogna calcolare un ranking probabilistico R che corrisponde a P. Per calcolare R da P si può utilizzare la BvN, che decompone una matrice doppiamente stocastica in una combinazione convessa di permutazioni di matrici.

In particolare data una matrice A doppiamente stocastica, esiste una decomposizione nella forma

$$\mathbf{A} = \theta_1 \mathbf{A_1} + \theta_2 \mathbf{A_2} + \cdots + \theta_n \mathbf{A_n}$$

Nel codice, la decomposizione viene calcolata attraverso l'utilizzo della libreria https://github.com/jfinkels/birkhoff. La matrice *P* ritornata al passo precedente dal metodo

findPropMatrix viene utilizzata all'interno del metodo *fairnessMethod* e attraverso il metodo *birkhoff_von_neumann_decomposition* della libreria viene calcolata la decomposizione.

4. Campionamento delle matrici di permutazione P_i con probabilità proporzionale a Θ_i e visualizza il corrispondente ranking r_i

Dato R, calcolato al passo precedente, si possono campionare i ranking *r* che adempiono ai vincoli di fairness.

Riprendendo il risultato della composizione abbiamo che i valori A_i sono le permutazioni delle matrici e corrispondono ai ranking r sull'insieme di item e i valori θ_i sono i coefficienti che corrispondono alla probabilità di campionare ogni ranking.

Questa parte viene svolta nell'ultima parte di codice nel metodo *fairnessMethod*. Il ranking prodotto viene poi utilizzato per riordinare il risultato unfair di partenza e produrre una versione che rispetti i vincoli di fairness utilizzati.

TEST EFFETTUATI E RISULTATI

Lanciata l'applicazione, la homepage presenta un casella di ricerca con la possibilità di selezionare attraverso una checkbox se attuare o meno un certo tipo di vincoli di fairness.
La sezione sottostante verrà utilizzata per visualizzare la DCG associata al ranking e appunto il ranking stesso, con titolo dell'articolo, pubblicazione e score associato all'item.

Segue un esempio di risultato per una query sottoposta al sistema nei seguenti casi:

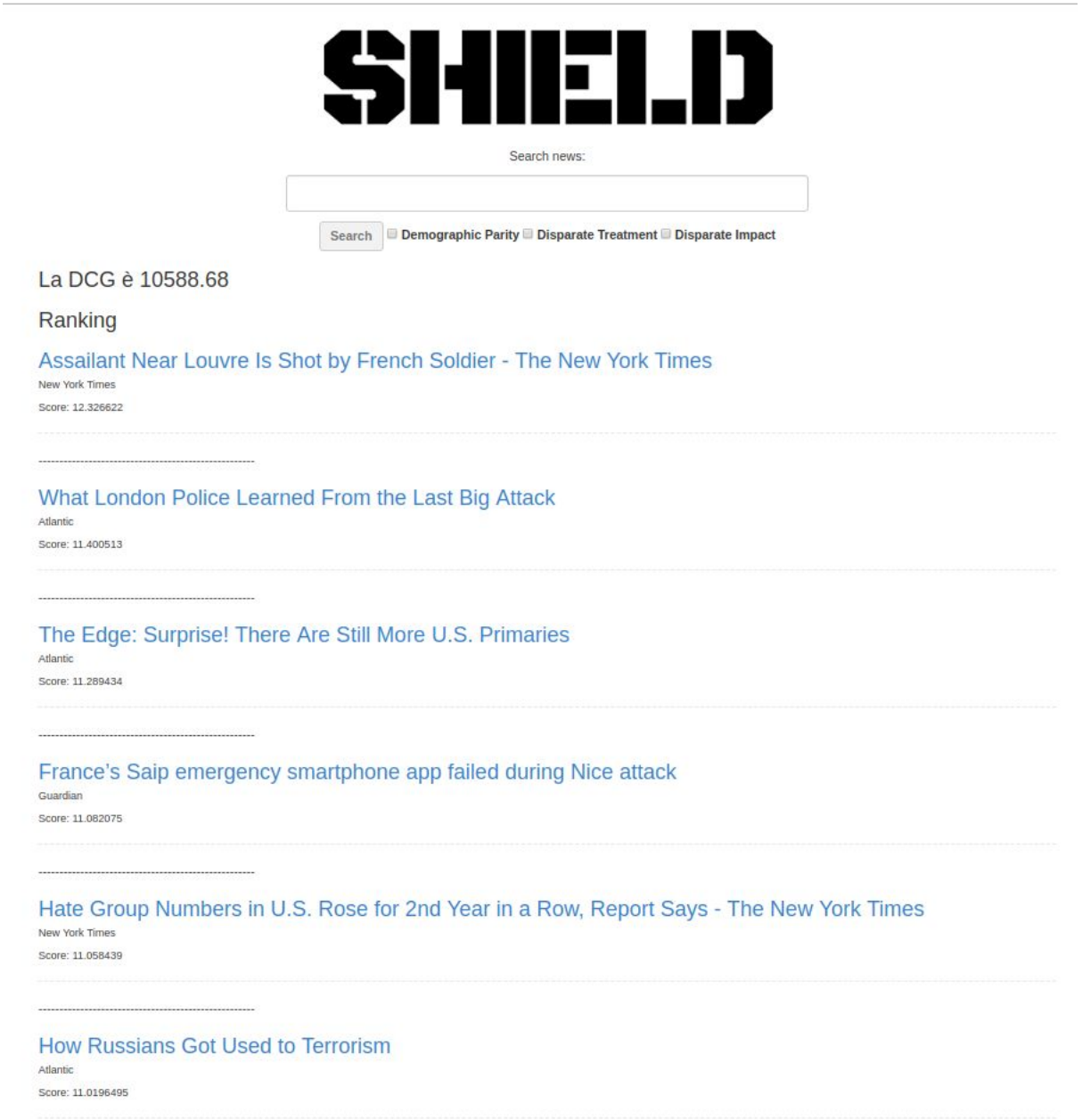
1. Ranking ottenuto nel caso unfair (no vincoli di fairness);
2. Ranking fair nel caso di vincoli di tipo "*Demographic Parity Contraints*";
3. Ranking fair nel caso di vincoli di tipo "*Disparate Treatment Contraints*";
4. Ranking fair nel caso di vincoli di tipo "*Disparate Impact Contraints*";

Inoltre viene visualizzata la *Discounted Cumulative Gain* associata al ranking e ne analizziamo il comportamente nei diversi casi, andando a vedere come questa cambia con l'applicazione dei diversi vincoli di fairness.
Il test seguente è stato effettuato a partire da un sottoinsieme dell'intero dataset di news, che è presente in forma di file csv nel repository:
https://github.com/Meyke/Progetto2BigDataFlask/blob/master/spark/articles_1.csv

Le motivazione di questa scelta sono dettate da alcune problematiche incontrate, descritte nella sezione trattante le conclusioni.

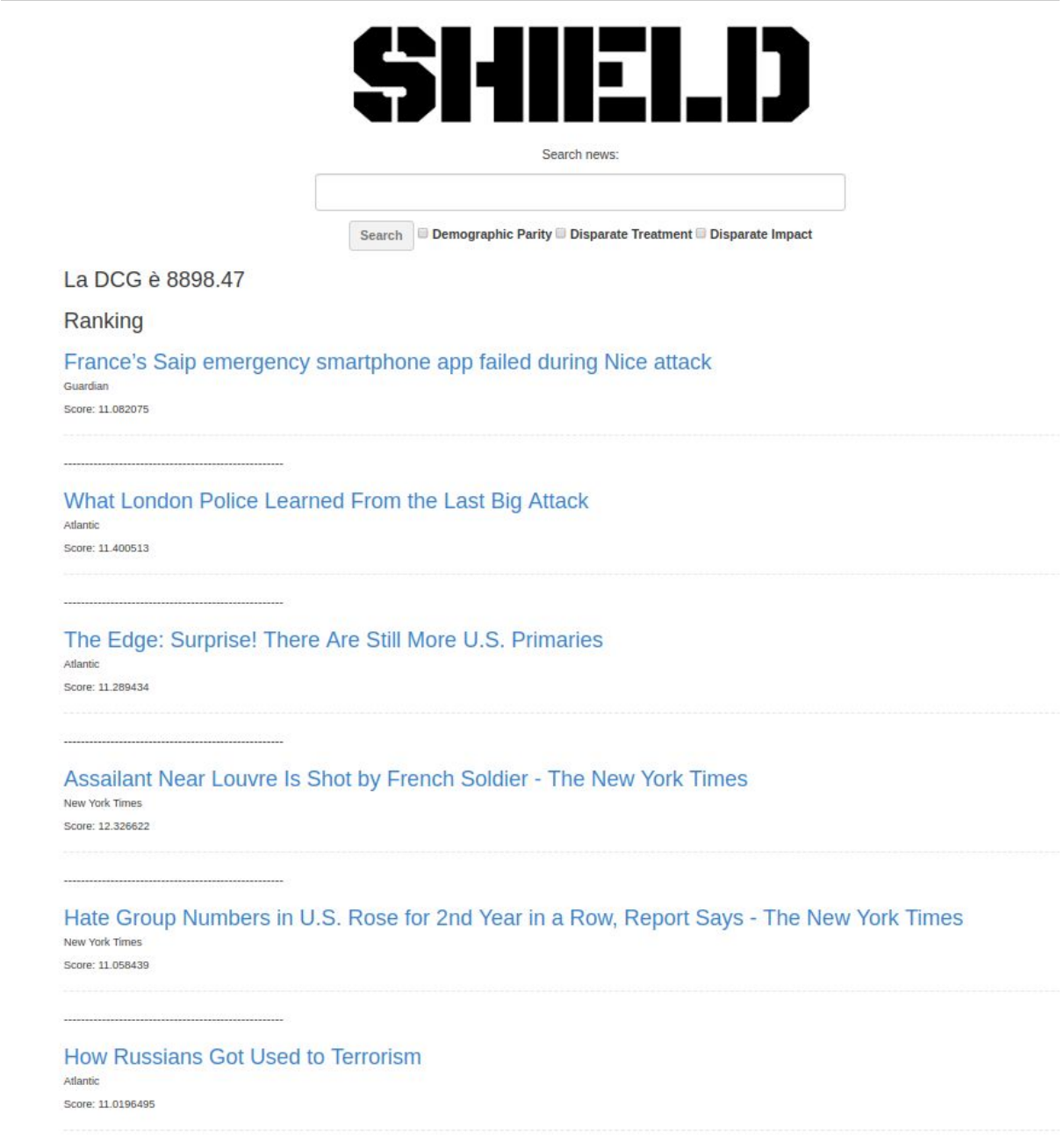
QUERY 1: *Terrorist attacks in 2016*

- *Esecuzione **senza** l'applicazione di vincoli di fairness*



Nell'immagine è raffigurato il risultato dell'esecuzione della query così come *Elasticsearch* lo restituisce. Gli score sono quelli calcolati da Elasticsearch e sono utilizzati senza normalizzazione per il calcolo del Discounted Cumulative Gain.

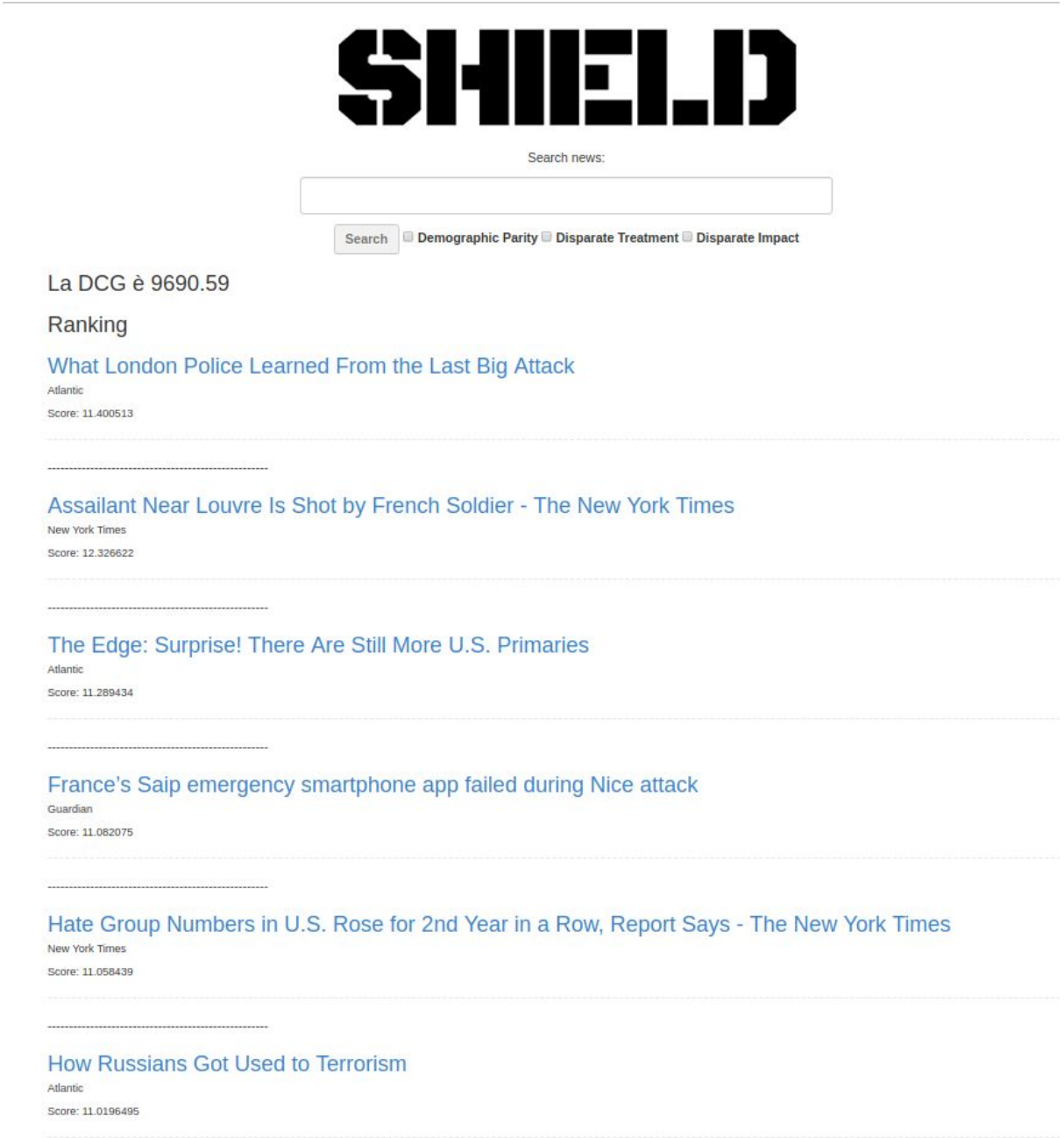
- *Esecuzione con l'applicazione di vincoli di tipo* ***Demographic Parity***



Nell'applicare il vincolo di *Demographic Parity*, possiamo notare un cambiamento importante nell'exposure del ranking. Come si può vedere, le gerarchie sono state riorganizzate in modo da garantire un'esposizione più equa e questo ha portato a far sì che documenti con score più basso ora risultano essere nelle prime posizioni del ranking. La DCG di questo ranking risulta essere assai più bassa di quella del caso precedente in

condizioni di unfairness. E questo perchè i vincoli di tipo "Demographic Parity" non prendono assolutamente in considerazione l'utilità associata ai diversi gruppi.

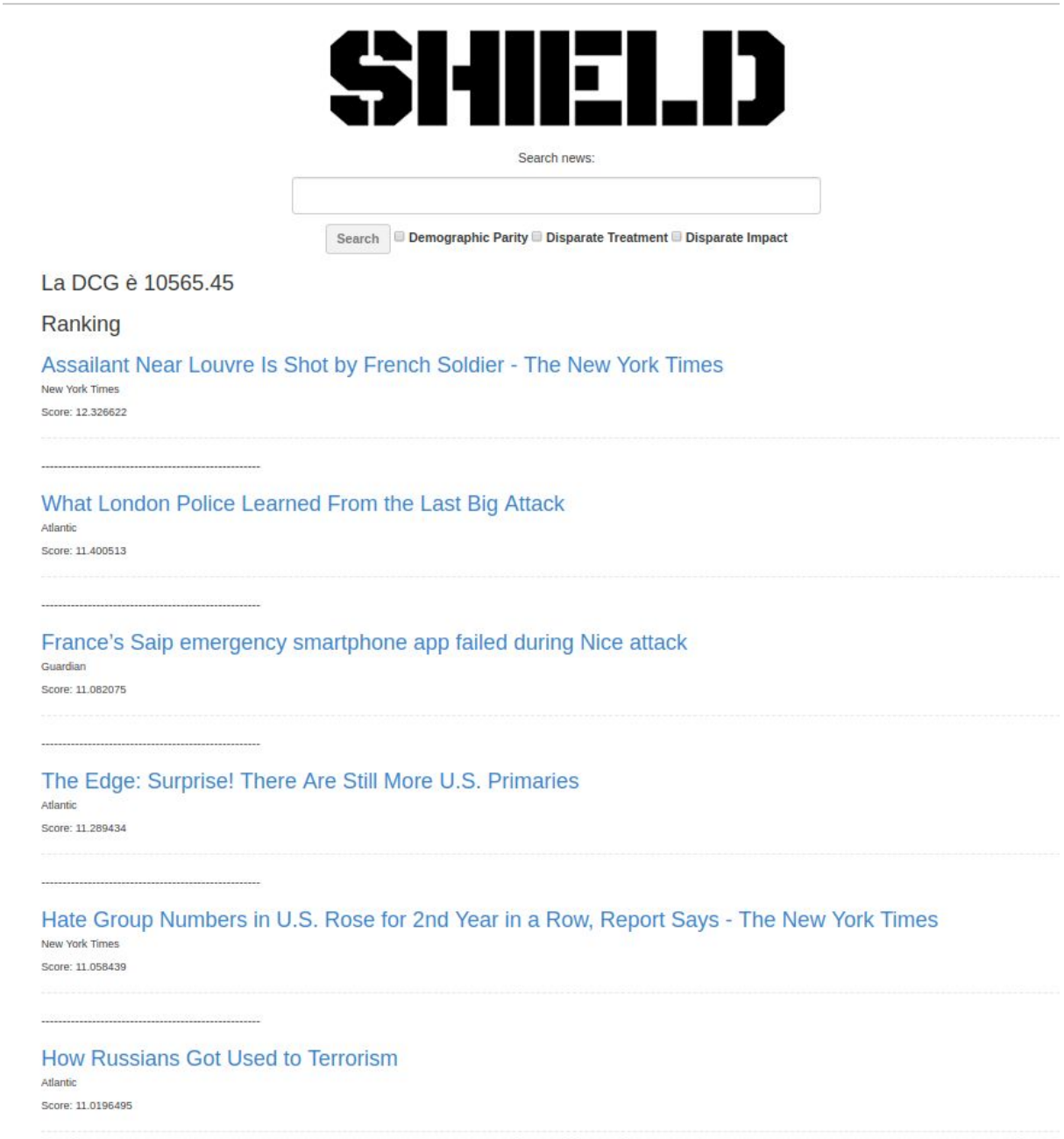
- *Esecuzione con l'applicazione di vincoli di tipo* ***Disparate Treatment***



Quando vengono applicati vincoli di tipo *Disparate Treatment,* si ottengono certamente dei risultati migliori dal punto di vista dell'utilità verso l'utente.

Ciò avviene perché, nel calcolo del ranking e del rimescolamento ottenuto nell'applicare i vincoli, si prendono in considerazione anche i valori di utilità associati ai diversi documenti. Questo permette di ottenere un ranking sicuramente più equo rispetto alla versione originale e con una DCG migliore rispetto al caso precedente, proprio perché l'utilità media dei diversi gruppi porta ad avere un risultato più simile a quello originale dove i documenti sono ordinati per rilevanza.

- *Esecuzione della query con l'applicazione di vincoli di tipo* ***Disparate Impact***



Come detto nelle sezioni precedenti, con i vincoli del tipo *Disparate Impact*, si prendono in considerazione l'utilità media per ogni gruppo e l'utilità del singolo documento in funzione di quello che sarà l'impatto futuro (es. un click su quel documento).
Tutto ciò ne risulta in un ranking diverso, non molto discostante da quello originale, con la news proveniente dal "Guardian" che scala verso l'alto, e per il resto identico.
Questo porta a una DCG molto vicina a quella del ranking originale (senza vincoli di fairness) e quindi in un risultato che, rispetto ai due casi precedenti, premia l'utilità verso l'utente.

PROBLEMATICHE RISCONTRATE

Nell'esempio mostrato, e nella maggior parte dei test, è stato utilizzato solo un sottoinsieme del dataset descritto. Questo sottoinsieme contiene circa 8000 record diversi provenienti da tre testate, che sono il "*New York Times*", il "*Guardian*" e "*Atlantic*".
Questa scelta è dovuta al fatto che se si utilizza l'intero dataset si hanno news provenienti da 15 testate diverse, e quindi i gruppi da prendere in considerazione non sono tre ma 15.
Per avere un risultato significativo al fine di vedere come lavorano queste tecniche di fairness non basta prendere i primi 6 risultati nel ranking, ma ne necessitano un po' di più (es. 20 o 25).
Inoltre, con dimensioni crescenti dei gruppi e degli elementi del vettore di rate u, si sono riscontrati molti problemi per il calcolo della matrice doppiamente stocastica. Infatti, risulta sempre più difficile ottenere una matrice cui somma degli elementi di ogni riga e somma degli elementi di ogni colonna dia *esattamente* 1 (in genere le librerie utilizzate forniscono valori prossimi a 1, ma non esattamente 1, non permettendo così di applicare la decomposizione di Birkhoff).
Non siamo gli unici ad aver riscontrato problemi nel calcolo della matrice doppiamente stocastica e della decomposizione di Birkhoff. Lo stesso problema è stato segnalato da altri utilizzatori infatti nella sezione issues nel repository della libreria utilizzata(libreria raccomandata proprio da chi ha scritto il paper). Ecco la sezione nel repository: https://github.com/jfinkels/birkhoff/issues

CONCLUSIONI E SVILUPPI FUTURI

Durante questo progetto sono state utilizzate tecniche e strumenti tipici nell'ambito dei big data come l'utilizzo di Spark e quello di Elasticsearch. Il codice presente nel repository è scritto per eseguire sia Spark che Elasticsearch sul singolo contenitore Docker e quindi con modalità pseudo-distribuita, ma test sono stati fatti anche utilizzando sia Spark che Elasticsearch su cluster Aws.

Inoltre sono state sperimentate le tecniche di fairness presenti nel paper di riferimento, attraverso la manipolazione dei dati e l'utilizzo di librerie utili per il calcolo matriciale e la risoluzione di problemi di ottimizzazione.

Si è cercato di applicare le tecniche su un dominio diverso da quelli usati per i test nel paper, e i risultati sono confortanti quando appunto si limita il risultato a pochi gruppi presenti e a ranking di dimensione relativamente piccoli, per i motivi citati nella sezione precedente.

Un possibile sviluppo futuro e forse il più importante, è quello di cercare un modo di determinare una matrice doppiamente stocastica in maniera esatta, superando i problemi incontrati. Fatto questo, il progetto potrebbe essere esteso in vari modi e in vari ambiti. Per esempio, nella rilevazione di diversi gruppi, magari non decisi a priori, ma con eventuali algoritmi già addestrati che riescono a determinare quali gruppi possono essere presenti in un certo corpus, oppure combinazioni di gruppi. Inoltre, si potrebbe fornire una sorta di explanation, che negli ultimi tempi va abbastanza di moda, del motivo per cui si siano ottenuti certi risultati. Ambiti ulteriori di applicazione può essere quello dei recsys, per esempio raccomandando prodotti analoghi ma non sempre di una certa marca, andando quindi ad intaccare il criterio di raccomandazione adoperato. Inoltre, queste tecniche le si potrebbero adoperare per applicare regole fair in dataset non molto strutturati, per esempio nella ricerca di ristoranti basandosi esclusivamente sulle recensioni, quindi testo non strutturato.

BIBLIOGRAFIA

Paper Fairness of Exposure in rankings: https://arxiv.org/abs/1802.07281

Elasticsearch documentation: https://www.elastic.co/guide/index.html

Elasticsearch API: https://www.elastic.co/guide/en/elasticsearch/reference/current/index.html

Elasticsearch on Docker: https://hub.docker.com/_/elasticsearch

Flask documentation: https://flask.palletsprojects.com/en/1.0.x/

Dataset from Kaggle: https://www.kaggle.com/snapcrack/all-the-news

Librerie python:

- Cvxpy: https://www.cvxpy.org/
- Numpy: https://numpy.org/
- Birkhoff: https://birkhoff.readthedocs.io/en/latest/ e https://github.com/jfinkels/birkhoff

SparkSQL: https://spark.apache.org/sql/