Information Retrieval
Progetto 1: Sistema di Raccomandazione per Tweet

Nicolosi Andrea (Mat. 834178)

Anno Accademico 2017/2018

Project Structure and Framework Used

The project is divided in two parts:

· Client: developed with Angular 5 and based on Clever Theme

· Server: developed with Spring Boot (based on Java 8).

System's information is stored into a MongoDB database, a NoSQL database document oriented, that stores data in flexible JSON-like documents

Tweet Recommender System - Client

- Full Text Search: The user can submit a query to the system, which will retrieve tweets that are relevant to query terms. A search type, exact match or similar match, must be selected and a category can be chosen as a filter for the query. "User preferences" flag may be used as a filter for the query instead of category selection.

- Recommendation: Based on user's preferences, the system submits a list of relevant tweets. User's preferences are obtained by monitoring user behavior.

Tweet Recommander System – Client Full Text Search Example

Tweet Recommender System – Client Recommendation Example

Tweet Liked

Gordon Ramsay

Can't wait to show you
https://t.co/wmX0U6D6eC

Fortnite

We just released a hotfix to make some adjustments to the spawn rate of the Ice Legion. Find out more here::
https://t.co/aAcE9P5WSP?

Tweet Recommanded

Gordon Ramsay

I'm finding a swimming pool of grease tonight on an all new @24HoursFOX as I head to #Memphis at 8/7c ! https://t.co/WdUjdh4D3r

♡ Like

Gordon Ramsay

East coast, let's take things for a spin on an all new #HellsKitchen in just under an hour !!!
https://t.co/CMwgLEldze

♡ Like

Gordon Ramsay

America, tonight black jackets are being awarded on an all new @HellsKitchenFOX at 9/8c ! https://t.co/bP2jtknDy8

♡ Like

How Full Text Search Work

Two types of search are supported:

- Exact Match: Uses the proximity operator to find all tweets that contain the query as written by user. Lucene PhraseQuery API is used to retrieve tweets from the index.

- Similar Match: Returns all tweets that contain all query terms or all tweets that contain terms that are similar to query terms. Results are ordered by their similarity to the query. Lucene FuzzyQuery API is used to retrieve tweets from index.

Full Text Search – Code Example

```java
@Override
public List<String> getTweetsTwitterIdFromIndex(String query, String category, LuceneConstants searchType) throws BaseException {
    String[] terms = query.split(" ");
    BooleanQuery.Builder builder = new BooleanQuery.Builder();

    switch(searchType) {
        case EXACT_MATCH: QueryBuilder.addPhraseQuery(builder, TweetToLuceneDocumentMapper.TEXT,terms);break;
        case SIMILAR_MATCH: QueryBuilder.addFuzzyQuery(builder, TweetToLuceneDocumentMapper.TEXT, terms, Occur.MUST);break;
        default: throw new InvalidSearchTypeException();
    }

    if(category != null) {
        QueryBuilder.addCategoryFilter(builder,category);
    }

    return getTwitteridFromIndexByQuery(builder.build(),query,TWEETS_LIMIT);
}
```

How Recommendation Work

User's behavior is monitored and some keywords, used for recommendation, are retrieved from full text search query and used to retrieve tweets. They are then filtered using categories of interest explicitly selected during registration or implicitly retrieved from tweets selected by the user.

All tweets that the user liked are removed from query results.

User's categories are always updated when user adds a like to a tweet of a cateogory that is not included inside his/her categories of interest.

Recommandation – Code Example

```java
public List<String> getRecommandedTweetsTwitterIdFromIndexByKeywords(String category,
        List<String> tweetsLiked, List<String> keywords) throws BaseException{

    BooleanQuery.Builder builder = new BooleanQuery.Builder();

    QueryBuilder.addCategoryFilter(builder, category);

    QueryBuilder.addLikedTweetsFilter(builder,tweetsLiked);

    QueryBuilder.addKeywordsFilter(builder, keywords);

    BooleanQuery booleanQuery = builder.build();

    return getTwitteridFromIndexByQuery(booleanQuery, "", TWEETS_LIMIT);
}
```

Tweets Crawling Process

Tweets are retrieved from twitter using Twitter4j API.

The process works in four steps:

1) All categories are retrieved from the categories.properties file;

2) For each category, all twitter users of that category are retrieved;

3) For each user, getUserTimeline(screen_name) is used to retrieve the last 20 tweets posted;

4) All tweets are saved inside the tweet collection. Each tweet is associated to a category that corresponds to the twitter user's category.

Tweets Crawling Process – Code Example

```java
@Override
public Integer retrieveTweetsFromUser(String screenName, String category) throws BaseException{
    try {
        Integer tweetsRetrieved = 0;

        try {
            this.saveTwitterUser(screenName, category);
        }catch(BaseException e) {
            Logger.info(e.getHrMessage());
        }

        List<Status> statuses = twitter.getUserTimeline(screenName);
        for (Status status : statuses) {
            try {
                this.saveTweet(status, category);
                tweetsRetrieved++;
            }catch(BaseException e) {
                Logger.info(e.getHrMessage());
            }

            if(status.getHashtagEntities().length > 0) {
                for(HashtagEntity hashtag : status.getHashtagEntities()) {
                    try{
                        this.saveHashtag(hashtag);
                    }catch(BaseException e) {
                        Logger.info(e.getHrMessage());
                    }
                }
            }
        }
        return tweetsRetrieved;
    } catch (TwitterException e) {
        throw new RetrieveTweetsException(e.getErrorMessage());
    }
}
```

Index Creation

Index is created using Lucene API

The process works in three steps:

1) All tweets are retrieved from the tweet collection;

2) Each tweet is converted to a Lucene Document;

3) All Lucene Documents are added to the index and saved inside the File System.

Index Creation – Code Example

```java
@Override
public String createIndex() throws BaseException {

    List<Tweet> tweets = (List<Tweet>) tweetRepository.findAll();

    Directory directory = luceneRepository.getIndexDirectory();

    IndexWriter indexWriter = getIndexWriter(directory);

    for(Tweet tweet : tweets) {
        Document doc = TweetToLuceneDocumentMapper.map(tweet);
        try {
            indexWriter.addDocument(doc);
        } catch (IOException e) {
            Logger.error(e.getMessage());
            throw new AddToIndexException(tweet.getTwitterid());
        }
    }


    try {
        indexWriter.close();
    } catch (IOException e) {
        Logger.error(e.getMessage());
        throw new IndexWriterCloseException();
    }

    return PropertyReader.getMessagesProperties(LUCENE_INDEX_SUCCESS);
}
```