Processing Natural Language for the Spotify API

Are sophisticated natural language processing algorithms necessary when processing language in a limited scope?

Patrik Karlström

Aron Strandberg
Processing Natural Language for the Spotify API

Bearbetning av Naturligt Språk till Spotifys API

Authors
Patrik Karlström | Aron Strandberg

Group 75

Supervisor
Michael Minock

Examiner
Örjan Ekeberg

2016-05-11
English Abstract

Knowing whether you can implement something complex in a simple way in your application is always of interest. A natural language interface is something that could theoretically be implemented in a lot of applications but the complexity of most natural language processing algorithms is a limiting factor.

The problem explored in this paper is whether a simpler algorithm that doesn’t make use of convoluted statistical models and machine learning can be good enough. We implemented two algorithms, one utilizing Spotify’s own search and one with a more accurate, offline search.

With the best precision we could muster being 81% at an average of 2.28 seconds per query this is not a viable solution for a complete and satisfactory user experience. Further work could push the performance into an acceptable range.
Swedish Abstract

Att kunna implementera till synes komplexa funktioner till sitt program på ett simpelt sätt är alltid av intresse. Ett gränssnitt för naturligt språk är något som teoretiskt sätt gör att implementera i många applikationer men komplexiteten i de flesta bearbetningalgoritmerna för naturligt språk är en begränsande faktor.

Problemet som presenteras är att utforska huruvida en simplare algoritm som inte använder sig av komplicerade statistiska modeller och maskininlärning kan vara tillräckligt bra. Vi implementerade två algoritmer, en som använder sig at Spotifys egna sök samt en med en noggrannare, lokal sökning.

För en komplett och tillfredsställande användarupplevelse är detta inte en duglig lösning. Den bästa precisionen hamnade på 81% och då med 2,28 sekunder som genomsnittligt tid per fråga. Fortsatt arbete kan förbättra prestandan till något mer acceptabelt.
## Contents

1 Introduction .......................................................... 5  
1.1 Problem statement .................................................. 5  
1.2 Purpose and Motivation ............................................. 6  

2 Background ............................................................ 7  
2.1 Natural Language Processing ....................................... 7  
2.1.1 Precision and Recall ............................................ 7  
2.1.2 The Gold Standard for Named Entity Recognition .......... 7  
2.2 The API ............................................................. 7  

3 Method ................................................................. 9  
3.1 Interacting with the Spotify API ................................... 9  
3.2 Application .......................................................... 9  
3.3 Corpus .............................................................. 9  
3.4 Algorithm .......................................................... 9  
3.4.1 Basic Algorithm ............................................... 9  
3.4.2 Extended Algorithm .......................................... 10  
3.5 Evaluating the results ............................................. 10  

4 Result ................................................................. 12  
4.1 Results With Entire Corpus ....................................... 12  
4.2 Results With Subsets of Corpus .................................. 12  
4.2.1 Without Queries for Albums .................................. 12  
4.2.2 Without Queries for Neither Albums nor Artists .......... 12  

5 Discussion ............................................................ 13  
5.1 General Discussion ................................................ 13  
5.2 Basic Algorithm .................................................. 13  
5.3 Extended Algorithm ............................................. 14  

6 Conclusion ........................................................... 15  

References .............................................................. 16  

A Corpus ............................................................... 17
1 Introduction

The current trend in technology as a whole is to move away from input methods that require the user to adapt to the computer and instead create I/O that adapts to the user. In other words you should be able to talk to your computer like you would another person.

A Natural Language Interface acts as interpreter between the user and the computer and allows users to read, write and modify data through the use of a natural language. A common application of an NLI is to let the computer listen via a microphone and process spoken language. This is very common in today’s smartphones, for example Google Now and Apple’s Siri. Another approach is to let the user enter the query into a text field and then the computer tries to parse the question or command as intuitively as possible.

An important aspect of processing natural language is to accurately determine what is the most likely wanted result given the input. In natural language there are usually more than one interpretation for every given statement, the correct one is usually derived from factors such as context and emphasis. These factors are difficult for a computer to process and handle correctly and thus the risk of being misunderstood is even greater when talking to a computer than it is talking to another human. The computer must therefore be able to guess what result the user would be most satisfied with, which isn’t a trivial thing to do.

Named entities are words or phrases that identifies an item, e.g a location or a person. An important subtask of Natural Language Processing is to reliably recognize and extract these named entities. Two main approaches to Named-entity Recognition are either rule-based NER, which relies on pre-made rules such as grammar, or machine-learning NER. A third approach is a hybrid between the two using only the strong points from each one (Mansouri et al. 2008).

Today a few main solutions exists for NLP. One of them is developed by the Stanford Natural Language Processing Group and their Stanford Named Entity Recognizer is based on a statistical model and use machine-learning to contextually extract entities (Stanford Natural Language Processing Group 2015). Another popular software is the Apache OpenNLP which is a hybrid; Using both grammatical rules and machine learning to process the input (Apache 2014).

Constructing an API call given an natural language string is not trivial. It depends heavily on the API itself. Different APIs can expect different queries. What every implementation has in common is that in the input string there will be a number of named entities. What exactly constitutes a named entity will vary. For example in a NLI for hotels "Imagine" would probably not be a named entity while in a NLI for a database of music tracks it would. A main focal point of finding suitable parameters is therefore NER.

1.1 Problem statement

A lot of NLP testings are done on huge datasets with thousands of sentences (Sang and Meulder 2003). When working with such huge datasets the application of statistical models has been very successful (MUC-7 2001). However, when processing natural language for an API call you rarely handle input that large. Instead you are looking at one or maybe two sentences.
The problem to be explored is whether it is beneficial to apply sophisticated grammatical rules and/or statistical models when parsing natural language into an API call rather than implementing an algorithm that simply checks all the reasonable combinations of the input. In such a small scope is the difference in performance even noticeable?

1.2 Purpose and Motivation

In an ideal world there would exist a general solution of NLP that could be implemented for any application and work with close to human-like accuracy while still being computationally fast. Although there are NERs with human-like accuracy (MUC-7 2001) the general case of NLPs are unfortunately not ideal.

Along with furthering the research of NLP, the algorithms are also becoming more advanced and require deeper and deeper understanding of statistical theories and models. Writing, customizing and maintaining such algorithms is not an easy task and is not a reasonable option for a lot of developers. Fortunately for a lot of developers they don’t really need to be able to parse thousands of sentences efficiently and accurately most of the times. Instead they would only need to parse one sentence. They would also have the benefit of instant feedback. If the Spotify has the track the search should be successful. In such a case there would be many benefits to a basic, essentially brute force, algorithm as long as it’s good enough.

The time investment, and thus the financial investment, of implementing the simpler algorithm is on a completely different scale compared to the more sophisticated ones. Especially if you wish to in any way customize the latter ones. Realistically not many employees would posses the necessary expertise to implement the more sophisticated ones and from a business perspective you would rather not rely on that one employee to maintain your code forever.

If there would be a way to make a simpler, more basic algorithm feasible given a limited context that would enable a great deal more flexibility for developers wishing to implement a simple NLI for their application. It would also make it easier for other developers to understand and maintain that code.
2 Background

2.1 Natural Language Processing

A natural language is a human language that has not been specifically constructed but rather evolved naturally through human interaction and socialization, in other words almost every spoken language (Lyons 1991). Natural Language Processing is the field of study that explores interaction between a computer and a such a language. The field of NLP is a very broad one that has a lot of major subtasks, some examples being optical character recognition (determine text on a picture) and machine translating (translating one human language to another). As previously stated, an important subtask of NLP that is of particular weight for our thesis is Named Entity Recognition, where in our case the named entities are songs, artists and albums.

2.1.1 Precision and Recall

One way to measure the accuracy of a pattern recognizing algorithm (such as named entity recognition) is to use precision and recall. Precision being the fraction of the retrieved material that is in fact relevant and recall being the proportion of relevant material actually retrieved (Rijsbergen 1979). For example if a program recognizes 3 organizations in a text containing 5 organizations but only 2 are correct (while one is a false positive) the precision is 2/3 and the recall is 2/5.

F1 score is the harmonic mean between precision and recall and is such calculated: $F_1 = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$ (Mansouri et al. 2008)

2.1.2 The Gold Standard for Named Entity Recognition

During the Message Understanding Conference 7 there was a Named Entity Recognition competition where a few research groups where up against each other as well as two human annotators. The groups were tasked with extracting names, organizations and locations from a dataset consisting of various news articles (MUC-7 1998). The best human $F_1$ score was 97.6% while the best computer got a score of 93.39% (MUC-7 2001). In the same competition the precision of the best computer was 92%. The gold standard of course being 100%.

2.2 The API

The API we have chosen to work with is libspotify. Libspotify is an API and SDK that allows developers to add Spotify music services to applications (Spotify n.d.). It allows us to, included but not limited to, search for tracks, artists, albums and playlists on Spotify as well as playing tracks and creating playlists.

Named entities in such an API are items such as tracks and artists. A query might be ”play Desolation Row”; Which is a pretty straight forward query. The named entity would then be the track ”Desolation Row”. Named entities in this setting might however be ambiguous. For example if the query is ”play Graceland” we have no reliable way of knowing whether the user meant the track or the album. We can only guess.
There may also be multiple artists with tracks that share a name. In such a case you would either have to specify which artist you are looking for or you could simply guess on the most popular track. Specifying an artist would require a keyword such as "by". For example "play Feel by Robbie Williams". That introduces the challenge of determining if something is a keyword or a named entity (or maybe not relevant at all). If I were to raise the query "play Me and Julio down by the schoolyard" it could be interpreted as play the track "Me and Julio down by the schoolyard" or play the track "Me and Julio down" by the artist "the schoolyard". In this case the latter interpretation doesn’t exist but that might not always be true.
3 Method

3.1 Interacting with the Spotify API

The connection to Spotify is handled through libspotify, a C library provided by Spotify.

libspotify has since been deprecated, being replaced by Android and iOS SDKs (Software Development Kit), but no new solution for playing Spotify programmatically on desktop computers has been provided. However, libspotify is still available, although no support is offered, but this has only caused minor headaches.

Instead of accessing libspotify directly, pyspotify (a Python wrapper for the library) is used. This makes the implementation considerably easier, since we can use Python instead of C. Python, being a much more high-level language, provides many abstractions that cut down on both difficulty of implementation and time spent on development.

3.2 Application

Our program is written in Python 3. Python was chosen because it is a simple language, with a sizable standard library (helping with ease of implementation), but first and foremost because it had one of the best libraries for interfacing with libspotify, as discussed in section 3.1.

3.3 Corpus

To decide which queries were relevant for the API, we came up with a basic corpus through introspection. The corpus consists of 69 different queries that test for tracks, artists and albums with varying degrees of accuracy. For example some queries asks for a song by a specific artist while some only asks for the song. However, the queries that doesn’t specify artists still expect the same URI as the ones that do (e.g "Play Desolation Row" and "Play Desolation Row by Bob Dylan"). Cover versions are considered incorrect. Some queries also asks for e.g "the album graceland" while others simply ask for "graceland" (refering to the album, not the song).

The complete corpus is included in appendix A and consists of 69 queries.

3.4 Algorithm

3.4.1 Basic Algorithm

Since the whole point of our algorithm is to explore whether it can be done easily the approach will be as greedy as possible while still maintaining a semblance of precision. It appears to be rare for people to list both track and album in the same query so in our implementation we try to find one track or album as well as one artist.

Our method uses a tokenizer to extract likely named entities based on the keywords. In addition to the tokenizer it also generates possibilities by splitting the query in two at every whitespace (one possibility for each split). The final list of possibilities is then the query split on each keyword as well as all combinations
of track/album - artist tuples given that the user writes everything in the correct order (e.g Kanye West instead of West Kanye). Which is a fair assumption.

When the possibilities have been generated Spotify is queried for each and every one of them and keep the results that are an exact match. Out of those result we pick that one that is most common across all possibilities and play that track.

### 3.4.2 Extended Algorithm

We extend our algorithm by implementing a local search instead of calling the API for every possible named entity. This means we only send one search to Spotify and thus the performance is not as limited to internet speed as the basic algorithm.

The dataset we use is the Million Song Dataset (Bertin-Mahieux et al. 2011). The entire dataset is roughly 280 GB in size, more than we had available during testing. In addition to the main dataset, several smaller files are provided. Because of if its good ratio of number of tracks to size, we used the subset called Track per Year, the subset for which information about release year was available. We then discarded the information about release year since it wasn’t relevant for our scope. This gave us a list of 469 189 pairs of artist and track name, more than enough to use as a basis for our research, but not enough for a real-world application.

We choose which result to play by determining how close it is to one of the possibilities. The closest result is the one that best matches our search. The search is performed in three levels:

1. **Exact matching.** If the query exactly matches an entity, it is reasonable to assume this is what the user was looking for.

2. **Substring match.** We first find all entities that has a substring that matches the query. Then, we perform a sequence matching on the results to find the best match. For example, a search for “Kanye” will yield the result ”Kanye West”.

3. **Sequence matching.** We calculate the Levenshtein distances between the query and all entries in the dataset. The Levenshtein distance is the number of characters that need to be either added, removed, or exchanged to transform one word into another. We then pick the entry with the closest distance. If multiple entries result in the same distance, the first one is picked.

### 3.5 Evaluating the results

To measure Precision & Recall (see section 2.1.1), we run a test suite where the result of each query is compared to its expected Spotify URI or, since some songs have multiple releases, expected URIs. The Spotify URI (Uniform Resource Identifier) is a unique string that identifies every resource on Spotify, in the format of `spotify:track:6rqhFgbbKwnb9MlUQDhG6`.

In our case precision will simply be the number of successful queries divided by the total number of queries. Since there is only one relevant answer to every query recall will be either 1 or 0. The arithmetic average recall of all the queries
will then be equal to the precision. This is turn leads to the $F_1$ score being equal to the precision. We will compare our $F_1$ score with the best performing computer at MUC-7 (see section 2.1.2) and of course to the gold standard of 100%.

We also measure the average execution time per query to better be able to compare our different methods to each other.
4 Result

Both algorithms were first tested on the complete corpus, which consists of 69 queries. The expected result for the queries are a mix of tracks, albums and artists. We then ran both algorithms again but this time excluded queries expecting an album as result and only asked for tracks and artists. We also did a third run of the algorithms with only the queries asking for tracks. Keep in mind that best performing computer at MUC-7 scored an $F_1$ score of 93,39%.

4.1 Results With Entire Corpus

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Basic</th>
<th>Extended</th>
</tr>
</thead>
<tbody>
<tr>
<td>$F_1$</td>
<td>40.58</td>
<td>55.07</td>
</tr>
<tr>
<td>Correct guesses</td>
<td>28</td>
<td>38</td>
</tr>
<tr>
<td>Incorrect guesses</td>
<td>41</td>
<td>31</td>
</tr>
<tr>
<td>avg seconds per query</td>
<td>0.29</td>
<td>3.38</td>
</tr>
</tbody>
</table>

Figure 1: Results when searching for tracks, albums, and artists

4.2 Results With Subsets of Corpus

4.2.1 Without Queries for Albums

The corpus in this case is consists of 51 queries. This time no queries expect an album as result.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Basic</th>
<th>Extended</th>
</tr>
</thead>
<tbody>
<tr>
<td>$F_1$</td>
<td>49.02</td>
<td>74.51</td>
</tr>
<tr>
<td>Correct guesses</td>
<td>25</td>
<td>38</td>
</tr>
<tr>
<td>Incorrect guesses</td>
<td>26</td>
<td>13</td>
</tr>
<tr>
<td>avg seconds per query</td>
<td>0.21</td>
<td>2.23</td>
</tr>
</tbody>
</table>

Figure 2: Results when searching for tracks and artists

4.2.2 Without Queries for Neither Albums nor Artists

The corpus in this case consists of 38 queries. All queries expect a track as result.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Basic</th>
<th>Extended</th>
</tr>
</thead>
<tbody>
<tr>
<td>$F_1$</td>
<td>61.16</td>
<td>81.58</td>
</tr>
<tr>
<td>Correct guesses</td>
<td>24</td>
<td>31</td>
</tr>
<tr>
<td>Incorrect guesses</td>
<td>14</td>
<td>7</td>
</tr>
<tr>
<td>avg seconds per query</td>
<td>0.33</td>
<td>2.28</td>
</tr>
</tbody>
</table>

Figure 3: Results when searching only for tracks
5 Discussion

5.1 General Discussion

There were a few problems with Spotify that affected the results. The main one was that Spotify’s search is not deterministic, meaning it doesn’t always return the same results for the same query (even when the two searches being run within just a few seconds of each other). This leads to the algorithm succeeding with a particular query during one test run but failing on the other. There were also a few times our connection with Spotify was lost or the Spotify library crashed, which lead to a few wasted test runs, but these have naturally been excluded from the results.

For the first queries (on the form ”play track” or ”play track by artist”), most misses came from the algorithm guessing on a cover version. The harder queries where when asked to play an album, particularly when the query asked for ”the album album” rather than just ”play album”.

Many of the mistakes made by the algorithms are a result of the ambiguity of the queries, as previously discussed. Without specifying if a query is meant to find a track, an album or an artist, there is no dependable way to determine this. It’s not uncommon for artists to have self-titled albums, or for albums to share their name with a track on the album.

As we had anticipated, a sizable portion of the negative results actually had the correct name, but the wrong type (for example, Until You Were Gone by The Chainsmokers yielded the album of the same name rather than the song as intended).

Since we experienced the most trouble with albums, we decided to run the tests excluding them. To satisfy our curiosity we also ran the test excluding both albums and artists. We suspect the most common queries will be for tracks only so it’s interesting to see how well we perform there.

Studying the difference between the results when including different types of queries yields some interesting conclusions. When excluding first queries for albums and then artists as well, the precision of both algorithms increased drastically. The basic algorithms for example, went from 49% to 61% when removing artists. However, while the percentage of correct answers increased by about 10 percentage points, the number of correct answers actually decreased by 1. The rest of the difference is actually made up by the removal of 12 incorrect answers, suggesting that the the algorithm did not perform particularly well on finding artists. When comparing all results in the same, one can conclude that both algorithms were considerably better at finding tracks than albums or artists.

5.2 Basic Algorithm

The basic algorithm consistently performed considerably worse in regards to accuracy, only managing to hit over 50% of queries in 1 of the 3 different test cases. As discussed above, many of these were the results of the inability to discern the type of results that is wanted.

However, the average running time for basic algorithm was lower than the extended by an order of magnitude. Despite this, the low percentage of correct results makes it unfeasible to use in any real applications.
5.3 Extended Algorithm

When we ran tests on the extended algorithm, we realized a significant portion of songs in our corpus was not featured in our dataset. To counter this, we modified the data to include them. This is the biggest weakness of the Extended Algorithm. Since the algorithm works by matching the user’s input against the dataset, anything not in the dataset cannot be found. If this method were to be used for a commercial application, the dataset would have to be kept up to date on a daily basis (preferably, even several times a day) to include new releases.

As stated in section 3.4.2, there dataset contains no information about albums whatsoever. Because of this, the extended algorithm is unable to get any positive results on these queries.

As mentioned above, the extended algorithm was on average about ten times slower than the basic algorithm. This can in part be attributed to our implementation being inefficient in its own right, but it is apparent that searching using a local database beforehand will consume more time. Since a query which finds no results will search the entire database, the extended algorithm will naturally spend a lot of time searching for albums, contributing to its high running time.
6 Conclusion

Looking at the results one can fairly quickly determine that for a complete NLI that supports queries for albums and artists (and if you want to be able to search for e.g playlists as well) our solution is not sufficient. The best precision we could produce was 55% with an average query time of 3.38 seconds. The fastest algorithm processed a query in 0.21 seconds but only had a precision of 40%. A NLI with a performance like that would obviously not improve the Spotify user experience.

When excluding albums both algorithms saw a performance boost. Especially the extended algorithm which achieved a 25 percentage points higher precision and each query took an average of one second less. While it is a significantly better performance it is still not enough for a satisfactory NLI.

Only querying for tracks yielded a 61% precision for the basic algorithm; Which is too low. The extended algorithm performed a precision of 81% which is considering the complexity behind the algorithm good enough. However, with an average of 2.28 seconds per query the solution as a whole is not good enough. Especially considering that you would have to continually update the offline dataset to maintain precision over time.

While we do not know how well a sophisticated algorithm would perform we can conclude that our non-sophisticated algorithm does not work good enough. Either it is too slow or it can’t find the sought after result well enough. Further work and optimizing could push the performance into an acceptable range.
References


A  Corpus

Tracks
play Desolation Row
play Dancing In The Dark
play Like A Rolling Stone
play Me And Julio Down by The Schoolyard
play I Wanna Dance With Somebody (Who Loves Me)
play Some Nights
play Sorry
play What Do You Mean?
play by The Way
play Tears In Heaven
play Call Me Maybe
play Sweet Lovin’
play Hero
play I Took A Pill In Ibiza
play Firestone
play ’74-’75
play Until You Were Gone
play Stand by Me
play Stan

Tracks with artist specified
play Desolation Row by Bob Dylan
play Dancing In The Dark by Bruce Springsteen
play Like A Rolling Stone by Bob Dylan
play Me And Julio Down by The Schoolyard by Paul Simon
play I Wanna Dance With Somebody by Whitney Houston
play Some Nights by Fun
play Sorry by Justin Bieber
play What Do You Mean? by Justin Bieber
play By The Way by Red Hot Chili Peppers
play Tears In Heaven by Eric Clapton
play Call Me Maybe by Carly Rae Jepsen
play Sweet Lovin’ by Sigala
play Hero by Family of The Year
play I Took A Pill In Ibiza by Mike Posner
play Firestone by Kygo
play ’74-’75 by The Connells
play Until You Were Gone by The Chain Smokers
play Stand by Me by Ben E. King
play Stan by Eminem

Albums
play the album Graceland
play Until Now
play the album Until Now
play the album The River
play the album The Dark Side of The Moon
play Modern Times
play the album Modern Times
play the album Born To Run
play the album 1
play Blonde On Blonde
play the album Blonde On Blonde
play Rumours
play the album Rumours
play the album Rumours by Fleetwood Mac
play By The Way
play the album By The Way
play the album The Marshall Mathers LP
play the album The Marshall Mathers LP

Artists
play Kanye West
play Paul Simon
play Bob Dylan
play Fleetwood Mac
play Daft Punk
play Rolling Stones
play Swedish House Mafia
play Alan Jackson
play The Beatles
play Buena Vista Social Club
play City of The Sun
play Leonard Cohen
play Eminem