Bringing algorithms and machine learning into library collections & services

Eric Lease Morgan <emorgan@nd.edu>
January 15, 2020

Abstract

Many aspects of librarianship have been automated to one degree or another. Now, in a time of "big data", is it possible to go beyond mere automation and towards the more intelligent use of computers; the use of algorithms and machine learning is an integral part of future library collection building and service provision. To make the point, this chapter first highlights a number of changes in librarianship that were deemed revolutionary in their time but are now taken for granted. Second, this essay compares & contrasts library automation with the exploitation of computer functionality. Finally, this chapter outlines both a number of possible machine learning applications for libraries as well as a few real world use cases. Librarianship is evolutionary, and the use of machine learning is a a part of librarianship's evolution. This chapter outlines how & why.

Seemingly revolutionary changes

At the time of their implementation, some changes in the practice of librarianship were deemed revolutionary, but now-a-days some of these same changes are deemed matter of fact. Take, for example, the idea of the catalog. For a long time a library catalog was merely an acquisitions list. Given some additional characteristics such as authors and subject headings, these acquisitions lists were morphed into books, books which could be mass produced and distributed. But the books were difficult to keep up to date, and they were expensive to print. As a consequence, catalog cards were invented, and the catalog became a massive set of drawers. Unfortunately, because the way catalog cards were produced, it is not feasible to assign more than three or four subject headings to any given book. If one does, then the number of catalog cards quickly gets out of hand.

At the same time, the idea of sharing catalog cards between libraries became common, and the Library of Congress facilitated much of the distribution. With the advent of computers the idea of sharing cataloging data as MARC (machine readable cataloging) became prevalent. The data structure of a MARC record is indicative of the time. Intended to be distributed on reel-to-reel tape, the MARC record is a sequencial data structure designed to be read from beginning to end, and all along there way there are checks & balances insuring the records' integrity. Despite the apparent flexibility of a digital data structure, the tradition of three for four subject headings per book still holds true. Now-a-days, the data from MARC records is used to fill databases, the databases' content is indexed, and items from the library collection are located by searching the index. The evolution of the venerable library catalog has spanned centuries, each evolutionary change solved some problems but created new ones. [HISTORY]

With the advent of the Internet, a host of other changes are (still) happening in libraries. Some of them are seen as revolutionary, and only time will tell whether or not these changes will persevere. Examples include but are not limited to:

- the advocacy of alt-metrics and open access publications
- the continuing dichotomy of the virtual library & library as place
- the creation & maintenance of institutional repositories
- the existence of digital scholarship centers
- the increasing tendency to license content

With the advent of the Internet, many of the traditional roles of libraries are not as important as they used to be. That does not mean the roles are unimportant, just not as important. Like many other professions, librarianship is exploring new ways to remain relevant when many of their core function are needed by fewer people.

Working smarter, not harder

The application of computer technology to the practice of librarianship has almost exclusively been about automation, and it has not exploited computer technology. Despite the fact that libraries have the world of knowledge at their fingertips, libraries do not operate very intelligently, where "intelligently" is an allusion to artificial intelligence.

Let's enumerate the core functionalities of computers. First of all, computers... compute. They are given some sort of input, assign the input to a variable, apply any number of functions to the variable, and output the result. This process -- computing -- is akin to solving simple algebraic equations such as the area of a circle or a distance traveled. There are two things of particular interest here. First, the input can be as simple as a number or a string (read "a word") or the input can be arbitrarily large combinations of both. Examples include:

- 42
- 1776
- xyzzy
- George Washington
- a MARC record
- the circulation history and academic characteristics of an individual
- the full text and bibliographic descriptions of all early American authors

What is really important is the possible scale of a computer's input. Libraries have not taken advantage of that scale. Imagine how librarianship would change if the profession actively used the full text its collections to enhance bibliographic description and resulting public service. Imagine how collection policies and patron needs could be better articulated if the totality of circulation histories and journal usage histories where thoroughly investigated in combination with patron characteristics and data from other libraries.

A second core functionality of computers are their ability to save, organize, and retrieve vast amounts of data. More specifically, computers save "data" -- mere numbers and strings. But when the data is given context, such as a number denoted as date or a string denoted as a name, then the data is transformed into information. An example might include the year 1972 and the name of Blake. Given additional information, which may be compared & contrasted with other information, knowledge can be created -- information put to use and understood. For example, Mary was born in 1951 and therefore she is 21 years older than Blake. Computers excel at saving, organizing, and retrieving data which leads to information and knowledge. The possibilities of computers dispensing wisdom -- knowledge of a timeless nature -- is left for another essay.

Like the scale of computer input, the library profession has not really exploited computers' ability to save, organize, and retrieve data. On the whole, the library profession does not understand the concept of a "data structure". Again, data becomes information when it is given context. In the world of MARC, when a string (one or more "words") is inserted into the 245 field of a MARC bibliographic record, then the string is denoted as a title. In this case, MARC is a "data structure" because different fields denote different contexts. There are fields for authors, subjects, notes, added entries, etc. This is all very well and good, especially considering when MARC was designed more than fifty years ago. But since then much more scalable, flexible, and efficient data structures have been designed.

Relational databases are a very good example. Relational databases build on a classic data structure known as the "table" -- a matrix of rows and columns where each row is a record and each column is a field. Think "spreadsheet". For example, each row may represent a book, and there are columns for authors, titles, dates, publishers, etc. The problem comes when a column may need to be repeatable. For example, a book may have multiple authors or more commonly, multiple subjects. In this case the idea of a table breaks down because it doesn't make sense to have a column named subject-01, subject-02, and subject-03. As soon as you do that, you will want subject-04. Relational databases solve this problem. The solution is to first add a "key" -- a unique value -- to each row. Next, for fields with multiple values, create a new table where one of the columns is the key from the first table and the other column is a value, in this case, a subject heading. There are now two tables and they can be "joined" through the use of the key. Given such a data structure it is possible to add as many subjects as desired to any bibliographic item.

But you say, "MARC can handle multiple subjects." True, MARC can handle multiple subjects, but underneath, MARC is a data structure designed when information was disseminated on tape. As such, it is a sequential data structure intended to be read from beginning to end. It is not a random access structure. What's more, the MARC data structure is really divided into three substructures: 1) the leader, which is always twenty-four characters long, 2) the directory, which denotes where each bibliographic field exists, and 3) the bibliographic section where the bibliographic information is actually stored. It gets more complicated. The first five characters of the leader is expected to be left-hand, zero-padded integer denoting the length of the record measured in bytes. A typical value may be 01999. Thus, the record is 1999 bytes long. Now, ask yourself, "What is the maximum size of MARC record?" Despite the fact that librarianship embraces the idea of MARC, very few librarians really and truely understand the structure of MARC data. MARC is a format for transmitting data from one place to another, not organization. As a whole, the library profession does not understand the concept of a relational database, despite the fact that it is a superior way of organizing data and turning thus turning it into information.

Moreover, there is more information in libraries besides bibliographic information. There is information about people and organizations. Information about resource usage. Information about licensing. Information about resources that are not bibliographic, such as images or data sets. Etc. When these types of information present themselves, libraries fall back to the use of simple tables, which are usually not amenable for turning data into information. There are many different data structures. XML became popular about twenty years ago. Since then JSON has become prevalent. More than twenty years ago the idea of Linked Data presented itself. All of these data structures have various strengths and weaknesses. None of them is perfect, and each addresses different needs, but they are all better than MARC when it comes to organizing data. Libraries understand the concept of manifesting data as information, but as a whole, libraries do not know how to manifest the concept using computer technology.

Finally, another core functionality of computers is networking and communication. The advent of the Internet is a relatively recent phenomenon, and the ubiquitous nature of computers combined with other "smart" devices has literally facilitated billions of connections between computers (and people). Consequently the data computed upon and stored in one place can be transmitted almost instantly to another place, and the transmission is an exact copy of the original. Again, like the process of computing and the process of storage, efficient computer communication builds upon itself with foreseen consequences. For example, who foresaw the demise of many centralized information authorities? For example, with the advent of the Internet there is less of a need/desire for travel agents, movie reviewers, or dare I say it, libraries.

Yet again, libraries use the Internet, but do they actually exploit it? How many librarians are able to create a file, put it on the Web, and share the resulting URL with whomever? Granted, centralized computing departments and networking administrators put up road blocks to doing such things, but the sharing of data and information is at the core of librarianship. Putting a file on the 'Net, even temporarily, is something every librarian ought to be able to know how (and be authorized) to do.

Despite the functionality of computers and their place in libraries over the past fifty to sixty years, computers have mostly been used to automate library tasks. MARC automated the process of printing catalog cards and eventually the creation of "discovery systems". Libraries have used computers to automate the process of lending materials between themselves as well as to local learners, teachers, and scholars. Libraries use computers to store, organize, preserve, and disseminate the gray literature of our time, and we call these systems "institutional repositories". In all of these cases, the automation has been a good thing because efficiencies were gained, but the use of computers has not gone far enough nor really evolved. Libraries have not truly exploited the functionality of computers. Lending and usage statistics are not routinely harvested nor organized for the purposes of monitoring nor predicting library patron needs/desires. The content of institutional repositories is usually born digital, but libraries have not exploited their full text nature nor created services going beyond rudimentary catalogs.

Libraries have not really embraced computer technology. Computers can do so much more for libraries than mere automation. While I will never say computers are "smart", their fundamental characteristics do appear intelligent, especially when used at scale. The scale of computing has significantly changed in the past ten years, and with this change the concept of "machine learning" has become more feasible. The following sections outlines how libraries can go beyond automation, embrace machine learning, and truly evolve its ideas of collections and services.

Machine learning: what it is, possibilities, and use cases

Machine learning is a computing process used to make decisions and predictions.

In the past, computer-aided decision-making and predictions where accomplished by articulating large sets of if-then statements and navigating down decision trees. The applications were extremely domain specific, and they weren't very scalable. Machine learning turns this process on its head. Instead of navigating down a tree, machine learning takes sets of previously made observations (think "decisions"), identifies patterns and anomalies in the observations, and saves the result as a mathematical model, which is really an n-dimensional array of vectors. Outside observations are then compared to the model and depending on the resulting similarities or differences, decisions or predictions are drawn.

Using such a process, there are really only four different types of machine learning: classification, clustering, regression, and dimension reduction. Classification is a supervised machine learning process used to subdivide a set of observations into smaller sets which have been previously articulated. For example, suppose you had a few categories of restaurants such as American, French, Italian, or Chinese. Given a set of previously classified menus, one could create a model defining each category and then classify new, unseen menus. The classic classification example is the filtering of email. "Is this message 'spam' or 'ham'?" This chapter's appendix walks a person through the creation of a simplified classification system. It classifies texts based on authorship.

Clustering is an unsupervised machine learning process which also creates smaller sets from a larger one, but clustering is not given a set of previously articulated categories. That is what makes it "unsupervised". Instead, the categories are created as an end result. Topic modeling is a popular example of clustering.

Regression predicts a numeric value based on sets of dependent variables. For example, given dependent variables like annual income, education level, size of family, age, gender, religion, and employment status, then one might predict how much money a person may spend on an an independent variable such as charity.

Sometimes the number of characteristics of each observation is very large. Many times some of these characteristics do not play a significant role in decision-making nor prediction. Dimension reduction is another machine learning process, and it is used to eliminate these less-then-useful characteristics from the observations thus making simplifying classification, clustering, or regression.

Some possible use cases

There are many possible ways to enhance library collections and services through the use of machine learning. I'm not necessarily advocating the implementation of any of the following ideas, but they are possibilities. Each is grouped into the broadest of library functional departments:

- **reference and public services**
    - given a set of grant proposals, suggest library resources to be used in support of the grants

    - given a set of licensed library resources and their usage, suggest other resources for use
    - given a set of previously checked out materials, suggest other materials to be checked out
    - given a set of reference interviews, create a chatbot to supplement reference services
    - given the full text of a set of desirable journal articles, create a search strategy to be applied against any number of bibliographic indexes; answer the proverbial question, "Can you help me find more like this one?"
    - given the full text of articles as well as their bibliographic descriptions, predict and describe the sorts of things a specific journal title accepts or whether a given draft is good enough for publication
    - given the full text of reading materials assigned in a class, suggest library resources to support them

- **technical services**
    - given a set of multimedia, enumerate characteristics of the media (number of faces, direction of angles, number and types of colors, etc.), and use the results to supplement bibliographic description
    - given a set of previously cataloged items, determine whether or not the cataloging can be improved
    - given full-text content harvested from just about anywhere, analyze the content in terms of natural language processing, and supplement bibliographic description

- **collections**
    - given circulation histories, articulate more refined circulation patterns, and use the results to refine collection development policies
    - given the full text of sets of theses and dissertations, predict where scholarship at your institution is growing, and use the results to more intelligently build your just-in-case collection; do the same thing with faculty publications

Implementing any of these possible use cases would necessarily be a collaborative effort. Implementation requires an array of expertise. Enumerated in no priority order, this expertise includes: subject/domain expertise (such as cataloging trends, circulation services, collection strategies, etc.), computer programming and data management skills (such as Python, R, relational databases, JSON, etc.), and statistical modeling (a understanding the strengths & weaknesses of different machine learning algorithms). The team would then need to:

1. articulate & share a common goal for the work
2. amass the data to model
3. employ a feature extraction process (lower case words, extract a value from a database, etc.)
4. vectorize the features
5. create & evaluate the resulting model
6. go to Step #2 until satisfied
7. put the model into practice
8. go to Step #1; this work is never done

For example, to model grant proposals to library resources, try this:

1. use classification to sub-divide each of your bibliographic index descriptions
2. apply the resulting model to the full text of the grants
3. return a percentage score denoting the strength of each resulting classification
4. recommend the use of zero or more bibliographic indexes

To predict scholarship, try this:

1. amass the full text and bibliographic descriptions of all theses and dissertations
2. topic model the full text
3. evaluate the resulting topics
4. go to Step #2 until satisfied
5. augment the model's matrix of vectors with bibliographic description
6. pivot the matrix on any of the given bibliographics

7. plot the results to see possible trends over time, trends within disciplines, etc.
8. use the results to make decisions

The GitHub repository accompanying this chapter describes how to do something very similar in method to the previous example. [GITHUB]

Some real-world use cases

Here at the University of Notre Dame's Navari Center for Digital Scholarship, we use machine learning in a number of ways. We cut our teeth on a system called Convocate. [CONVOCATE] In this case we obtained a set of literature on the theme of human rights. Half of the set was written by researchers in non-governmental organizations. The other half was written by theologians. While both sets were on the same theme, the language of each was different. An excellent example was the use of the word "child". In the former case, children were included in documents about fathers and mothers. In the later case, children often referred to the "Children of God". Consequently, queries referring to children were often misleading. To rectify this problem, a set of broad theme were articulated, such as Actors, Harms and Violations, Rights and Freedoms, and Principles and Values. We then used topic modeling to subdivide all of the paragraphs of all of the documents into smaller and smaller sets of paragraphs. We then compared the resulting topics to the broad themes, and when we found correlations between the two we classified the paragraphs accordingly. In retrospect, this process was not ideal, but we were learning and the resulting index is useful.

On a regular basis we find ourselves using a program called Topic Modeling Tool, which is a GUI/desktop application heavily based on the venerable MALLET suite of software. [TOOL, MALLET] Given a set of plain text files and an an integer, Topic Modeling Tool will create a weighted lists of latent themes found in a corpus. Each theme is really a list of words which tend to cluster around each other, and these clusters are generated through the use of an algorithm called LDA (Latent Dirichlet Allocation). When it comes to topic modeling, there is no such thing as the correct number of topics. Just as in the traditional process of denoting what a corpus is about, there can be many distinct topics or there can be a few. Moreover, some of the topics may be large and others may be small. When using a topic modeler, it is important to iteratively configure and re-configure the input until the results seem to make sense.

Just like every other machine learning application, Topic Modeling Tool bases its "reasoning" on a matrix of vectors. Each row represents a document, and each column is a topic. At the intersection of a document row and a topic column is a score denoting how much the given document is "about" the calculated topic. It is then possible to sum each topic column and output a pie chart illustrating not only what the topics are, but how much of the corpus is about each topic. Such can be very insightful.

By adding metadata to the matrix of vectors, even more insights can be garnered. Suppose you have a set of plain text files. Suppose also you know the names of the authors of each file. You can then do topic modeling against your corpus, and when the modeling is complete you can add a new column to the matrix and call it authors. Next, you update the values in the authors column with author names. Finally, you "pivot" the matrix on the authors column to calculate the degree each authors' works are "about" the calculated topics. This too can be quite insightful. Suppose you have works by authors A, B, C, and D. Suppose you have calculated topics I, II, III, and IV. By updating the matrix and pivoting the results, you might discover that author A discusses topic I almost exclusively, whereas author B discusses topics I, II, III, and IV in equal parts. This process works for just about any type of metadata: gender, genre, extent, dates, language, etc. Whats more, Topic Modeling Tool makes this process almost trivial. To learn how, see the GitHub repository accompanying this chapter. [GITHUB]

We have used classification techniques in at least a couple of ways. One project required the classification of press releases. Some press releases are deemed mandatory -- declared necessarily to publish. Other press releases are considered discretionary -- published at the will of a company. The domain expert needed a set of 100,000 press releases classified into either mandatory or discretionary piles. We used a process very similar to the process outlined in this chapter's Appendix. In the end, the domain expert believe the classification process was 86% correct, and this was good enough for them. In another project, we tried to identify articles about a particular yeast (Cryptococcus neoformans), despite the fact that the articles never mentioned the given yeast. This project failed because we we unable generate an accuracy score greater than 70%. This was deemed not good enough.

We are developing a high performance computing system called the Distant Reader, and it uses machine learning to do natural language processing against an arbitrarily large volume of text. Given one or more documents of just about any number or type, the Distant Reader will:

1. amass the documents
2. convert the documents into plain text
3. do rudimentary counts & tabulations against the plain text
4. calculate statistically significant keywords against the plain text
5. extract narrative summaries against the plain text
6. use Spacy (a natural language processing library) to classify each & every feature of each & every sentence into parts-of-speech and/or named entities [SPACY]
7. save the results of Steps #1 through #6 as plain text and tab-delimited files
8. distill the tab-delimited files into an SQLite database
9. create both narrative as well as tabular reports against the database
10. create an archive (.zip file) of everything
11. return the archive to the student, researcher, or scholar

The student, researcher, or scholar can then analyze the contents of the .zip file to get a better understanding of its contents. This analysis ("reading") ranges from perusing the narrative reports, to using desktop tools to visualize the data, to exploiting command-line tools to investigate the data, to writing software which uses the data as input. The Distant Reader scales everything between a single scholarly report, hundreds of book-length documents, and thousands of journal articles. Its purpose it to supplement the traditional reading process, and it uses machine learning techniques at the its core.

Summary and conclusion

Computers and libraries are a natural fit. They both excel at the collection, organization, and dissemination of data, information, and knowledge. Compared to most professions, computers have been used in the practice of librarianship for a very long time. But, for the most part, the functionality of computers in libraries has not been fully exploited. Advances in machine learning coupled with the data/information found in libraries, represents an opportunity for both librarianship as well as the people for whom librarianship serve. Machine learning can be used to enhance library collections and services. The idea may seem too expensive or revolutionary to implement, and only tell whether or not such is true.

Notes and links

[HISTORY] For quick and easy-to-read histories of library catalogs and MARC see ["Catalogs, their history" and "MARC for library use"].

[CONVOCATE] https://convocate.nd.edu

[TOOL] https://github.com/senderle/topic-modeling-tool

[MALLET] http://mallet.cs.umass.edu

[GITHUB] https://github.com/ericleasemorgan/bringing-algorithms

[SPACY] https://spacy.io

Appendix: Train and classify

This appendix lists two Python programs. The first (train.py) creates a model for the classification of plain text files. The second (classify.py) uses the output of the first to classify other plain text files. For your convenience, the scripts and some sample data ought to be available in a GitHub repository. [GITHUB] The purpose of including these two scripts is to help demystify the process of machine learning.

Train

The following Python script is a simplistic classification training application.

Given a file name and a list of directories containing .txt files, this script first reads all the files' contents and the names of their directories into sets of datum and labels (think "categories"). It then divides the data and labels into training and testing sets. Such is a best practice for these types of programs so the models can be evaluated for accuracy. Next, the script counts & tabulates ("vectorizes") the training data and creates a model using a variation of the Naive Bayes algorithm. The script then vectorizes the test data, uses the model the classify the test data, and compares the resulting classifications to the originally supplied labels. The result is an accuracy score, and generally speaking, a score greater than 75% is on the road to success. A score of 50% is no better than flipping a coin. Finally, the model is saved to a file for later use.

```
# train.py - given a file name and a list of directories containing
# .txt files, create a model for classifying similar items

# require the libraries/modules that will do the work
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection         import train_test_split
from sklearn.naive_bayes             import MultinomialNB
import glob, os, pickle, sys

# sanity check; make sure the program has been given input
if len( sys.argv ) < 4 :
  sys.stderr.write( 'Usage: ' + sys.argv[ 0 ] + " <model> <directory> <another directory>
[<another directory> ...]\n" )
  quit()

# get the name of the file where the model will be saved
model = sys.argv[ 1 ]

# get the rest of the input, the names of directories to process
directories = []
for i in range( 2, len( sys.argv ) ) : directories.append( sys.argv[ i ] )

# initialize the data to analyze and its associated labels
data   = []
labels = []

# loop through each given directory
for directory in directories :
  # find all the text files and get the directory's name
  files = glob.glob( directory + "/*.txt" )
  label = os.path.basename( directory )

  # process each file
  for file in files :

    # open the file
    with open( file, 'r' ) as handle :

      # add the contents of the file to the data
      data.append( handle.read() )

      # update the list of labels
      labels.append( label )

# divide the data/labels into training sets and testing sets; a best practice
data_train, data_test, labels_train, labels_test = train_test_split( data, labels )

# initialize a vectorizer, and then count/tabulate the training data
vectorizer = CountVectorizer( stop_words='english' )
data_train = vectorizer.fit_transform( data_train )

# initialize a classification model, and then use Naive Bayes to create a model
classifier = MultinomialNB()
classifier.fit( data_train, labels_train )

# count/tabulate the test data, and use the model to classify it
data_test       = vectorizer.transform( data_test )
classifications = classifier.predict( data_test )

# begin to test for accuracy
count = 0

# loop through each test classification
for i in range( len( classifications ) ) :

  # increment, conditionally
  if classifications[ i ] == labels_test[ i ] : count += 1

# calculate and output the accuracy score;
# above 75% begins to achieve success
print ( "Accuracy: %s%% \n" % ( int( ( count * 1.0 ) / len( classifications ) * 100 ) ) )

# save the vectorizer and the classifier (the model)
# for future use, and done
with open( model, 'wb' ) as handle : pickle.dump( ( vectorizer, classifier ), handle )
```

```
exit()
```

Classify

The following Python script is a simplistic classification program.

Given the model created by the previous script (train.py) and a directory containing a set of .txt files, this script will output a suggested label ("classification") and a file name for each file in the given directory. This script automatically classifies a set of plain text files.

```
# classify.py - given a previously saved classification model and a directory of .txt
files, classify a set of documents

# require the libraries/modules that will do the work
import glob, os, pickle, sys

# sanity check; make sure the program has been given input
if len( sys.argv ) != 3 :
  sys.stderr.write( 'Usage: ' + sys.argv[ 0 ] + " <model> <directory>\n" )
  quit()

# get input; get the model to read and the directory containing the .txt files
model     = sys.argv[ 1 ]
directory = sys.argv[ 2 ]

# read the model
with open( model, 'rb' ) as handle : ( vectorizer, classifier ) = pickle.load( handle )

# process each .txt file
for file in glob.glob( directory + "/*.txt" ) :

  # open, read, and classify the file
  with open( file, 'r' ) as handle : classification =
classifier.predict( vectorizer.transform( [ handle.read() ] ) )

  # output the classification and the file's name
  print( "\t".join( ( classification[ 0 ], os.path.basename( file ) ) ) )

# done
exit()
```