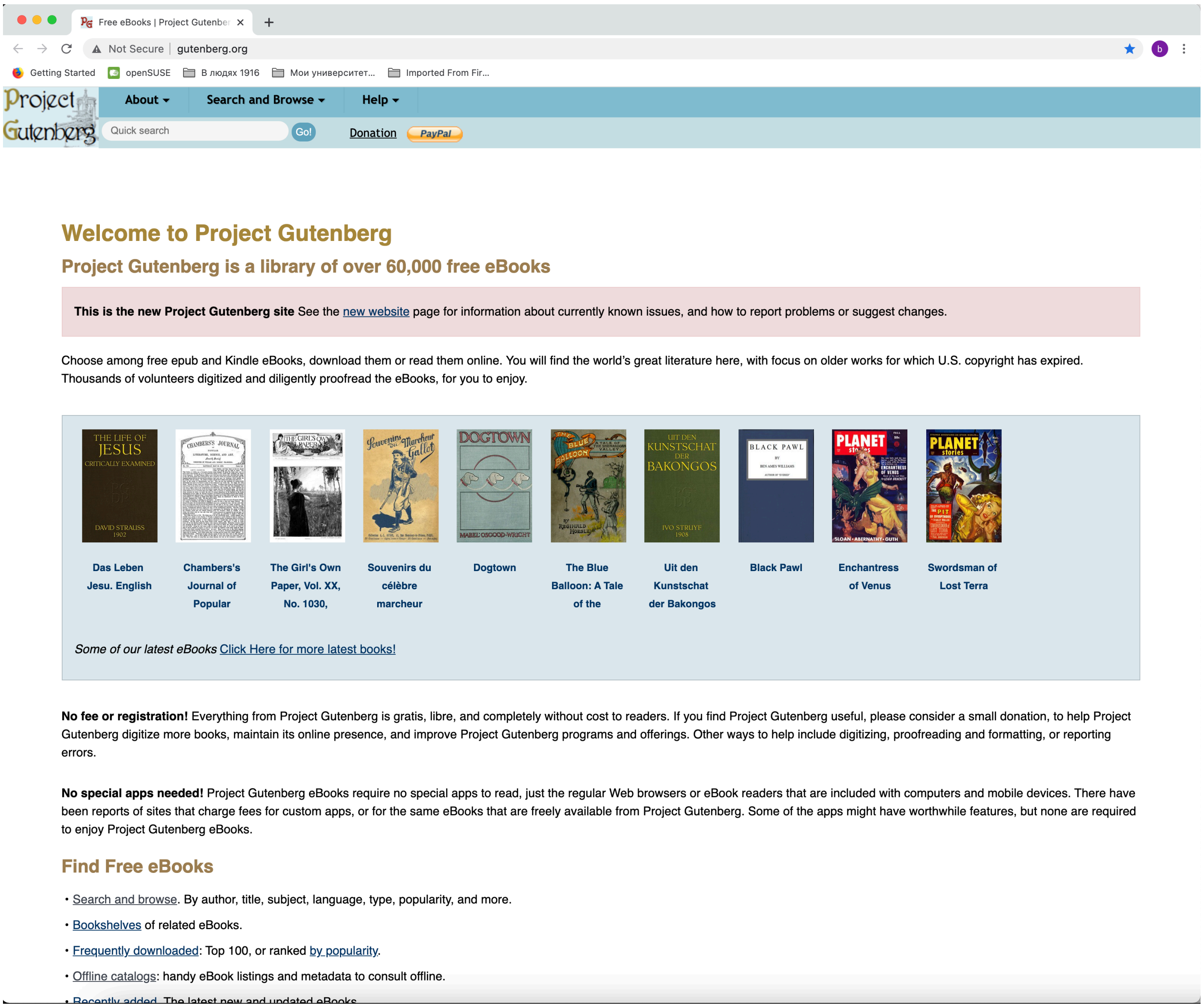
#Group# Hummus — Moocho

> *Gutenberg Poetry Corpus Building with Pig + Hadoop + Python (English-only)*



Welcome to Project Gutenberg

Project Gutenberg is a library of over 60,000 free eBooks

This is the new Project Gutenberg site See the new website page for information about currently known issues, and how to report problems or suggest changes.

Choose among free epub and Kindle eBooks, download them or read them online. You will find the world's great literature here, with focus on older works for which U.S. copyright has expired. Thousands of volunteers digitized and diligently proofread the eBooks, for you to enjoy.

Das Leben Jesu. English · Chambers's Journal of Popular · The Girl's Own Paper, Vol. XX, No. 1030, · Souvenirs du célèbre marcheur · Dogtown · The Blue Balloon: A Tale of the · Uit den Kunstschat der Bakongos · Black Pawl · Enchantress of Venus · Swordsman of Lost Terra

Some of our latest eBooks Click Here for more latest books!

No fee or registration! Everything from Project Gutenberg is gratis, libre, and completely without cost to readers. If you find Project Gutenberg useful, please consider a small donation, to help Project Gutenberg digitize more books, maintain its online presence, and improve Project Gutenberg programs and offerings. Other ways to help include digitizing, proofreading and formatting, or reporting errors.

No special apps needed! Project Gutenberg eBooks require no special apps to read, just the regular Web browsers or eBook readers that are included with computers and mobile devices. There have been reports of sites that charge fees for custom apps, or for the same eBooks that are freely available from Project Gutenberg. Some of the apps might have worthwhile features, but none are required to enjoy Project Gutenberg eBooks.

Find Free eBooks

- Search and browse. By author, title, subject, language, type, popularity, and more.
- Bookshelves of related eBooks.
- Frequently downloaded: Top 100, or ranked by popularity.
- Offline catalogs: handy eBook listings and metadata to consult offline.
- Recently added. The latest new and updated eBooks.

Raw Data

1. aprrish_json -> poetry

book_id, line

```
…
{ “book_id”:  “19”, “line”: “O’ Captain! My Captain!” }
…
{ “book_id”: “41322”, “line”: “If we shadows have offended,” }
…
```

info:

- 3,000,000+ lines of verses
- composed by:
-
    - 1500+ authors -> potential number of labels
    - 1000+ books total

2. index.all -> index

> *It's text data, obtained from official site of Gutenberg Project. Contains the catalogue of all books published till 2020-11-11.*

```
A Key into the Language of America, or an Help to the Language
  63701
 of the Natives in that part of America Called New-England,
 by Roger Williams
 [Subtitle: Together with Briefe Observations of the Customes,
  Manners, and Worships, &c. of the Aforesaid Natives, etc.]
```

Reconstructure the Two Files and Crude Cleaning

Step 1. Prepare Index from PG into JSON Format

> *Python*

```
{"author": "Bryce Walton", "book_id": "63709", "title": "Prisoner of the Brain-Mistress"}
{"author": "Bill Wesley", "book_id": "63707", "title": "The Primus Curse"}
{"author": "Various", "book_id": "63706", "title": "Graham's Magazine, Vol. 18, No. 4, April 1841"}
{"author": "Colonel Prentiss Ingraham", "book_id": "63705", "title": "Buffalo Bill's Bold Play"}
```

note:

- since the `index.all` is pure text file, so it's hard to extract all useful information completely, and about **2%** of them are lost during this process.

Step 2. Use `JsonLoader()` and `JsonStorage()` to Load and Store Them

> *Pig*

```
a = load 'poetry_json' using JsonLoader();
b = load 'index_json' using JsonLoader();
...
```

Step 3. Join Poetry and Index to Build the Complete Corpus into One Single JSON File

> ***Pig Scripting** & a little bit of **Python***

1. **Pig** for `JOIN` and `GROUP` operation, also appending new attributes to corpus.
2. **Python** for fixing the problem raised from *step1(data loss)*.
3. author, num_books, num_lines, book::book_id, book::title, book::lines.

   3.1 `author` will be the **label** for final training of model.

   3.2 `num_lines` is for optimizing the selecting of **testing dataset**.

   3.3. `num_books` is the number of books contained in by the author.

   3.4 `book` is a bag type containing the basic **book info** and poetries.

   3.5 `book::lines` is a tuple in `book` contains all **lines** of poetries.

```
{"author":"Various","num_books":70,"num_lines":19006,"book":[{"book_id":"38438","title":"The Melody of Earth","lines":[{"line":"move the white peacock as though through the "},{"line":"to the american tract societi for seed and the philosoph "},{"line":"garden john oxenham ...
...
```

Step 4. Cleaning and Stemming

> ***Pig UDF in Python***

```
register moof.py using streaming_python as moocho;
a = foreach corpus generate moocho.stem(line) ...;
```

```
# moof.py
class PorterStemmer{
    ...
}
stem()
```

- have to code the `PorterStemmer()` out.
- poetry is **word-sensitive**, so stemming may affect the classification and

clustering in a significant manner, and due to that, two sets of corpus is built, one stemmed, one just cleaned. in the end, I will compare results of these two different sets.

Step 5. Selecting Datasets for Training and Testing

> *Pig*

- using the above-mentioned attribute: `num_lines`, make sure the dataset chosen for the correspondsing author(label) isn't too small.

Structuring Text Datasets

Calculating TF-IDF

> *Hadoop Mapreduce*

```
#phase1
-------
mapper:  ((word, author), 1)
reducer: ((word, author), word_count)

#phase2
-------
mapper:  (author, (word, word_count))
reducer: ((word, author), (word_count, words_of_author))

#phase3
-------
mapper:  (word, (author, word_count, words_of_author, 1))
reducer: ((word, author), TF-IDF)
```

Model Training

> *Python `sklearn`, `nltk`, etc.*

atrribute	number
original volume	3,000,000
original authors	1,192
cleaned volume	~2,500,000
cleaned authors	428
(1st) Alighieri Dante	109,443
most others	~2,000

There're more than 3 million lines of verses by 1,192 different authors in my original data, after cleaning process, I narrowed it down to 2.5 million lines by 428 authors, which means available number of potential labels is 428. Among all these authors survived cleaning, I got a very unevenly distributed corpus. In which the top author Alighieri Dante has more than 100,000 lines, while most of the others are lingering under 2,000.

Normally, sentiment analysis of `positive` and `negtive` needs > 10000 of data, but in my situation, only the top two of them satisfy the threshold. So I cannot do traditional classification, for I could get pretty miserable reults. My data is not large enough. I cannot do multi-classification, because that means temendous feature engineerings should be done in the first place, which I have no time for. The final compromise is, I do the classification between only two authors, e.g., select Whitman and Yeats, and do classification between just these two.

I tried a few algorithms first, K-Means, Nieve Bayes, Random Forest, and SVM. But they all perpormmed pretty poorly, the accuracy for most of them was lower than 70%, the best of them—SVM, just over 81% on 9:1 proportion of training to testing datasets. Then I turned to regression algorithms, everything started to be bright. The fact that regression algorithms performs finer than clasification algorithms and clustering algorithms may indicate the discrete character of my datasets. The final algorithm I decided to use was logistic Regression, implemented by `sklearn` package of Python.

author	year	nationality	number of lines
Dante Alighieri	1265-1361	Italy	109,443
John Milton	1608-1674	British	37,284
W. B. Yeats	1865-1939	Irish	14,097
Walt Whitman	1819-1892	America	9,334
William Shakespeare	1564-1616	British	7,885
Ezra Pound	1885-1972	America	1,854
T. S. Eliot	1888-1965	America / British	1,710

I select seven authors which generates 21 pairs for model training. The authors' range from 13th century to 19th century, from 1,710 lines of T. S. Eliot to 109,443 lines of Alighieri Dante, from Irish to America of nationality. The first table shows the features of selected authors and the second table shows the training results of accuracy.

author1	**author2**	**9:1**	**7:3**	**5:5**	**3:7**	**1:9**
dante	milton	0.8718191825289056	0.9114817433462956	0.9256813740615036	0.9328128128128128	0.9376926821353238
dante	yeats	0.9465172274189768	0.9399742248502767	0.8901032477292035	0.9560110548134501	0.918809443254013
dante	whitman	0.9597583970811424	0.9527786231663522	0.9697749196141479	0.9544440455431327	0.9690056639395846
dante	shakespeare	0.9736454652532391	0.9326170096521581	0.9327475642766264	0.9752186588921283	0.9424246737841044
dante	pound	0.9854091456077015	0.9843480623151911	0.9840270658191511	0.9861237012760616	0.9858744394618834
dante	eliot	0.9866342138945425	0.9887578626183301	0.9950025848698949	0.9930187098575817	0.9940529622980251
milton	yeats	0.9122654621264767	0.9188996070025008	0.8918702195904269	0.891427112911974	0.8638249887235002
milton	whitman	0.9425267955101696	0.93536	0.948785046728972	0.8960129038762035	0.9411128729998558
milton	shakespeare	0.9245762711864407	0.9373988127361036	0.9275025799793601	0.9260198066493751	0.8457171405213916
milton	pound	0.952572949781603	0.9701899327144196	0.9732894255397	0.9527004098253674	0.9677859847309291
milton	eliot	0.9879288437102922	0.9798253893345917	0.9840177580466148	0.9555735478478897	0.9801538079880923
yeats	whitman	0.8510258697591436	0.7128153023810576	0.7865437692027287	0.7978549252046289	0.8354717639870821
yeats	shakespeare	0.8310007227807346	0.8156004676799733	0.8717370286819207	0.8691647477496337	0.847003712214955
yeats	pound	0.9053311793214863	0.8983890954151177	0.8962497424273645	0.8890192914252182	0.9093625498007968
yeats	eliot	0.9260827092152394	0.944792973651192	0.9130884635281945	0.9024695908588278	0.9244288224956063
whitman	shakespeare	0.9115077857956704	0.8889960226587923	0.8868753432180121	0.8488163020677255	0.9061391541609822
whitman	pound	0.9021262234222073	0.8725622057834567	0.9020004495392223	0.8864880864288885	0.9032006245120999
whitman	eliot	0.8962035024926499	0.9008844587954513	0.924297520661157	0.9037719037719038	0.9238057639163048
shakespeare	pound	0.9031737493275955	0.8554232133806386	0.8594171613599568	0.8760385510136258	0.9042441248684672
shakespeare	eliot	0.9453293155402497	0.9738430583501007	0.8593075987169561	0.7465024013363959	0.9294330518697226
pound	eliot	0.7967599410898379	0.7055572612833896	0.7099447513812155	0.7949656750572083	0.7554309740714786

Several interesting characteristics emerged from this results table, to sum it up:

1. The testing accuracy is proportional to the natural log of the difference between number of lines of `author1` and number of lines of `author2`.

   $$TestAccuracy = C_0 ln(numLines(author1) - numLines(author2))$$

   The general aspect of coefficient C_0 I haven't digged into details on the complete corpus. Because doing that will require 91,370 pairs of training, that's about 127 days of my CPU calculation—day and night. But with the given 21 pairs of testing results on Logistic Regression model, this proportional rule describes its feature so seamlessly. I think it might be something going on there worth better generalization. In our current case, C_0 is approximately 0.326 .

2. There's no direct relations between the training-testing ratio and testing accuracy of the model. e.g., the **dante-milton** pair shows that with the increasing of training proportion, the testing accuracy increases; but for the **whitman-shakespeare** pair, the trend goes in other way; while most of the

pairs perform best on 9:1 ratio, and then the trend start to tremble. I think the stirring factor may be the unevenness of my data, since I didn't apply proportional sampling when selecting testing datasets, just naive random selection, but that's just a hypothetical conjectureture, haven't checked out.

3. The highest testing accuracy can reach a level of more than 99.5% of **dante-eliot** pair on 5:5 training proportion, and lower than 71% of **pound-eliot** pair on 7:3 traning proportion. You see, Dante is from 13th century Italy, while Eliot and Pound are both from 19th century America, dispite the influence of volume variance, the result is quite reasonable. Because Eliot shares fewer commonalities with Dante than with Ezra Pound, which natually results in more difficulties of distinguishing from each other. Similar intuitive phenomena are demonstrated in the table, but that as well merely signs that the model works fine.

Discussion

The analysis of the sample 21 pairs shows that it is practical to apply traditional Machine Learning methods on poetry corpus for distinction between two authors whose collection of lines are more than 1,710, and the naive accuracy can reach at least 70%. The process which I show here, does not involve many feature engineering work (TF-IDF is the only feature considered). I believe that with proper forefront featuring work, the model's average accuracy can be raised up to an acceptable level. And for the time being, I haven't been able to make the most of every piece of the corpus, just a small fragment of it. But the practice is proved to be expandable, minal modifications of the code is gurentted for that objective, you can customize your own pairs, no matter 21 or 42 or 63 pairs. But the number iterations of total training reaches its limit very quickly, so thourough ananlysis is very time-consuming, and Hadoop Cluster is recommended for that sort of tasks.

Acknowledgements

This project is composed by two phases, first phase for preparing data for use of model training, which I used Hadoop Mapreduce, Apache Pig; the second phase I used sklearn of Python for model training and evaluating. Raw data comes from Allison Parish and Project Gutenberg.

References

Python Machine Learning Book by Sebastian Raschka

Project Gutenberg

Gutenberg Poetry Corpus by Allison Parrish

Arthur M. Jacobs (2018). The Gutenberg English Poetry Corpus: Exemplary Quatitative Narrative Analyses, published by Frontiers in Digital Humanities

TF-IDF-HadoopMapReduce