The Embedded Spur

By Josh Bevan

Abstract

The Embedded Spur is a data visualization project inspired by the increasing aptitude that intelligent algorithms offer for abstracting and understanding large datasets. As a greater proportion of data expands in scale beyond the point of comprehensibility by human laypersons, and is increasingly designed for automated creation and use by, for, from, and between machines - algorithms capable of translating this data into higher-level abstractions that are closer in scale and mode to the human layperson's grasp are becoming incredibly important. The Embedded Spur is simply a case study in the transvergence of software aesthetics, the technical application of statistical 'learning' algorithms, and the humanitarian concern of the evolution of ideas.

Concept

The data for the Embedded Spur (ES) comes from the Seattle Public Library's (SPL) Library Collection Inventory, where the Title, ISBN and Publishing Year of each book in the collection was pulled. I used Python to process the data further.

```
import sys
import isbntools
import isbnlib
import csv
import numpy as np
from sklearn.manifold import TSNE
from sklearn.feature_extraction.text import CountVectorizer

import nltk
from nltk.corpus import stopwords

from sklearn.cluster import KMeans
```

```
from sklearn.decomposition import PCA
from sklearn.preprocessing import scale
from sklearn.decomposition import TruncatedSVD
```

The Imports

I used isbnlib to use the ISBN of each book to pull its description from Google Books.

In [221]:
```
# read the data and pull out ISBNs
ISBNs = []
publishingYears = []
with open('foo.csv', encoding = 'utf-8', newline='') as csvfile:
    rowReader = csv.reader(csvfile, delimiter=',', quotechar='"')
    for row in rowReader:
        ISBNs.append((row[4],row[6]))
        publishingYears.append(row[2])
    ISBNs = ISBNs[1:]    # chop off the header
    ISBNs = [s for s in ISBNs if s[1] != '']
    publishingYears = publishingYears[1:]
    publishingYears = [p for p in publishingYears if p != '']

print(len(ISBNs))
```

In [230]:
```
# use isbnlib to pull descriptions for each book
metas = []
i = 0
failedCount = 0
for j, t in enumerate(ISBNs):
    clean = isbnlib.to_isbn13(isbnlib.clean(t[1]))
    try:
        metas.append((t[0], isbnlib.desc(clean), publishingYears[j]))
        print("success ", clean)
    except:
        print("Failed: ", clean)
        metas.append((t[0], t[0], publishingYears[j]))
        #print("Fail count at ", failedCount)
        failedCount += 1
        next
    i += 1
```

I then used NLTK to tokenize the descriptions (clean non-standard formatting - casefold - distinguish into words) and remove stop words (non-semantically interesting words).

Then I used SKLearn to vectorize the cleaned descriptions into a shared high dimensional space where each dimension corresponds to the count of a unique word.

The Euclidean distance between the points that each description holds in this high dimensional space can be thought of as the difference between their diction - their word choice. This is an imperfect measure of a description's related topic - as it is assumed that similar topics would share the use of certain words.

In [233]:
```
# do a word-to-vector
vectorizer = CountVectorizer()
CountVectorizer()
X = vectorizer.fit_transform([d for _, d, p in betterMetas])
```

I then used the SKLearn implementation of the KMeans clustering algorithm. When specified a somewhat-arbitrary K number of clusters, the algorithm will iteratively learn where to place the 'mean' of each cluster. The algorithm includes a classification of the provided data into these clusters. In more abstract terms this can be seen as clustering book descriptions together that are nearby in diction - approximating a 'topic' classification - though machine generated.

```
In [234]:  # perform high dimensional clustering
           kmeans = KMeans(n_clusters=20, random_state=0).fit(X)
```

I then calculate and store the deviation of each class prediction, which is the euclidean distance between each description and its predicted class's mean point. This deviance will become the meat and bones of what information we have to gain from the visualization.

```
In [235]:  # collect the error of each description from its cluster mean
           errors = []
           for i in range(len(X.toarray())):
               foo = np.linalg.norm(kmeans.cluster_centers_[kmeans.labels_[i]] - X.toarray()[i])
               errors.append(foo)
```

I store each book's title, publishing year, predicted class, and class deviance into a CSV to be brought into Processing.

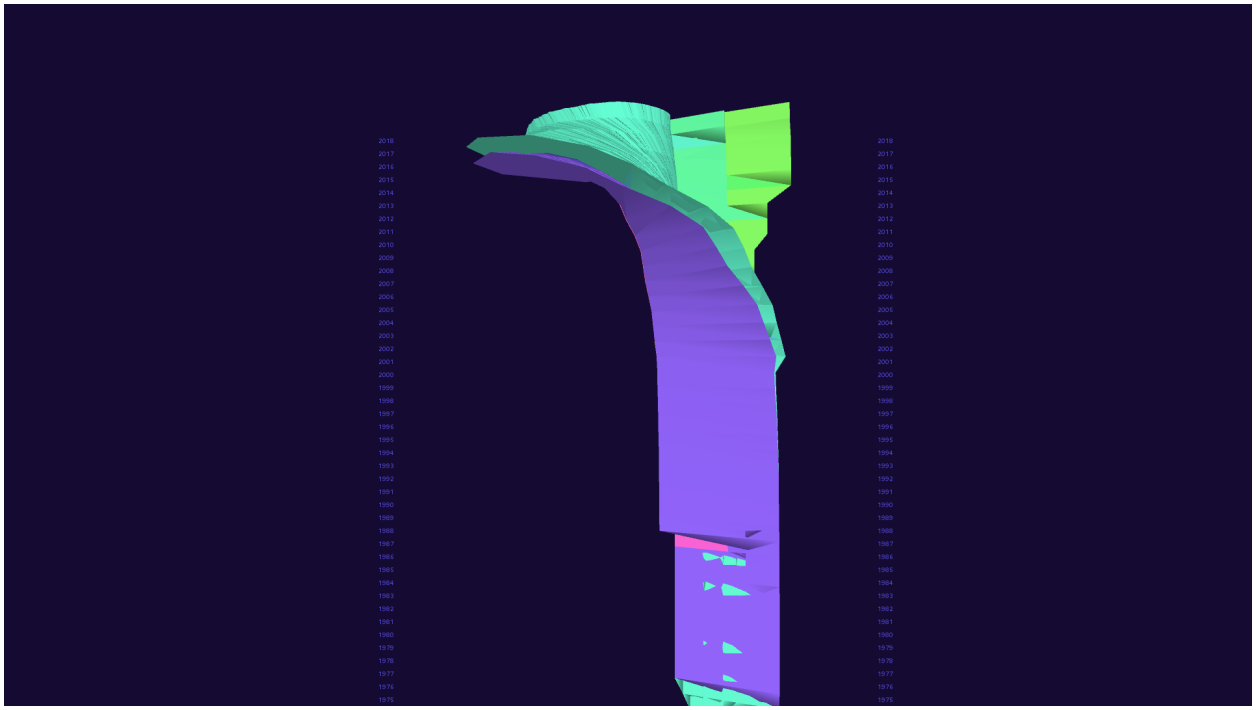
In Processing, I store each title, year, class, and deviance as properties in an array of [Book], then generate an array of [Topic] for each predictable class, each filled with an array of [Book] objects that relate to that [Topic].

I then sort each topic's collection of books by their deviance (called Error in the program). Then, I generate an array of [Year] objects relating to each possible year of publication, each filled with an array of [Topic] objects that are the subset of their parent [Topic] object that contain only books published on or before that publishing year.

Then each year is allowed to generate a 'slice', which is a collection of [BookParticle] objects that hold a color associated to each book's associated [Topic], their title, and their position in three dimensional space - which is calculated by taking the [Book]'s deviance as the radius around an origin, and its index among all books in the slice as the angle about the origin.

Then the 3 Dimensional mesh is generated as a collection of Quad_Strip panels by cycling through the years two at a time, connecting each [BookParticle] to itself as it appears in the next year slice, and then to the next [BookParticle] pair around the radius.

The result is shown here:



How to Read It

You can read the rate at which the shape coils upwards as the rate of publications in a particular topic - twistier is speedier publication. And you can read the density of the vertical lines within one topic ‘leaf’ as the density of descriptions at a certain variance away from the topic mean. If this density increases or decreases with time, then that reveals an evolution in the variance in diction of topics through the years.