Fall 2015 S675 “Statistical Learning and High Dimensional Data” Final Project Report

Miao Chen, Dec 17 2015

1. Introduction

For this project we were offered data from a corpus and topic modeling resulted, and I applied statistical methods on the data for deriving analysis results. The analysis centers around these two main research inquiries: can dimensionality reduction on high-dimensional data help text OCR correction (mainly using the corpus data)? Can dimensionality reduction help better represent text documents? I used the Darwin corpus data to address the first inquiry, and mainly used the Darwin topic modeling data to tackle the second inquiry.

I made the first inquiry about OCR correction because the original corpus comes from a digital library where books are stored in images and images are digitized to text via OCR techniques. This is where the raw text data for the corpus and for the topic modeling are from. It is common that OCR transformation usually introduces OCR errors, and as can be inspected in the vocabulary of the Darwin corpus, there are quite a number of OCR errors by eyeballing the vocabulary. Therefore, I examined two different vector spaces, one with dimensions reduced, the other not, to see which space can better help with OCR correction. The second inquiry mainly uses topic modeling data, trying to recover the original structure through document similarity.

The methods involved are truncated SVD (a similar approach to principal component analysis), classical multidimensional scaling (CMDS) and K-means clustering, and measuring metrics involved are cosine similarity (similarity), arccosine distance (dissimilarity), and KL Divergence.

2. Data Set

The data I used have two parts: corpus data and topic modeling data. We were first offered topic modeling data resulted from the corpus data, and then the corpus data as well. The corpus data is actually a document-word matrix along with a vocabulary list of the corpus. There are 593 documents and 204,562 in total.

The data set comes from Colin Allen’s and Jaimie Murdock’s research project, in which they first collect raw text data from different resources, and then preprocess them into word-document matrix (corpus.dat as shown below), and then perform topic modeling based on word-document matrix using Latent Direchlet Allocation to produce document-topic matrix (topics data as shown below).

2.1 Data Content

The data contain these content respectively:

- Corpus data
    - corpus.dat (document-word matrix stored in a file, 593 rows * 204,562 columns, in sparse representation; the cell value is the frequency count of a word in a document)

    - vocab.txt (document containing vocabulary of the corpus, 204,562 words/lines)
- Topic modeling data
    - Topics (the topic documents contains document-topic matrix, listing the probability of topics on each document; topic numbers = 20, 40, 60, or 80, and therefore there are 4 such document-topic matrix files)
    - KL Divergence (document similarity are calculated using KL-di

2.2 OCR Errors

Unfortunately, the raw text data was not clean – it contains a number of OCR errors. The reason is because the raw text was not generated digitally, but rather digitized via OCR technique from scanned books from research libraries, and thus OCR errors are almost inevitably introduced in the OCR process. Below lists several example OCR errors that I detected from the vocabulary by eyeballing and also lists their correct spellings:

Word with OCR error	Correct form
hetter	better
hetween	between
himselt	himself
historj	history
historv	history
hlue	blue
hlqh	high

OCR errors introduces noise to data processing, especially in the context of high-dimensional data set. It brings in more dimensions than the actual number of dimensions. For example, there is no word “hlgh” but it exists in the Darwin corpus vocabulary because of OCR error, and thus adds to one more dimension to the data. The vocabulary contains 204,562 in total, of which a large number of the words can come from OCR errors.

The OCR error words motivates the first inquiry of this project: how we can map these words to a space where we can easily find its correct spelling, i.e. in a space the OCR error word is close to its correct word. We will use the original document-word space and principal component space to answer this question and examine the distance resulted from these spaces.

The implication is that, if we find a particular space in which an OCR error word is close to its correct form, for example, finding “hetter” is close to “better”, then we can correct “hetter” to “better” easily, and thus this constructed space provides a basis for further OCR error correction.

3. Constructing Vector Space for Words and Analyzing Word Similarity

I constructed two types of space: one directly from the document-word matrix from the corpus.dat, and the other by applying truncated SVD to projecting document-word matrix to a space of principal components. Then in these two vector spaces, I obtain the cosine similarity between OCR words, and also between OCR words and their correct forms, and then analyze the effects of the two spaces on drawing close OCR words and their correct forms.

*This part of work is performed in Python as it provides friendly packages and methods to preprocess large sparse matrix as well as to analyze sparse matrix.

3.1 Constructing Vector Space for Words

3.1.1 Using the Word-Document Space

Starting from the word-document matrix from the corpus.dat file, for each word we can obtain a vector by using the information of in which document this word occurs and the word frequency in its occurring documents. Given a word, knowing its column number (suppose it's column k) in the word-document matrix, its vector can be obtained by extracting the column k from the matrix.

For example, for word "historv", we can find its matrix by this way: first, we know it is the 86129th column in the matrix, and this column vector records the information of the frequency that "historv" occurs in each document (row). Then we use this vector as the vector representation for "historv".

	Word1	…	"historv"	…	Word204562
Doc1			0		
Doc2			0		
…			…		
Doc593			0		

Table. Obtaining vector for word "historv" (the shaded area).

The shaded column in the above table provides vector information for word "historv". We thus obtain a vector composed of frequency in documents for each words, and then we are ready to compute word similarities based on the vectors.

Another way of constructing vector for words based on document-word matrix is to build a word co-occurrence matrix, and then using co-occurring words as vector dimensions. But since the resulted co-occurrence matrix will be extremely huge, 204562*204562, I did not use this approach, but just used the vector using documents as dimensions for representing words. This simplistic approach results in a 593-dimension vector for each word, as opposed to dimensions if generated from word co-occurrence matrix. The assumption is, if two words are both present in a same set of documents, or similar sets of documents, then they are similar or relevant to each other, with similarity of 1 or nearly to 1; and if they are not present together in any documents, then they are not similar to each other, with similarity of 0.

3.1.2 Constructing Word-PrincipalComponent Space

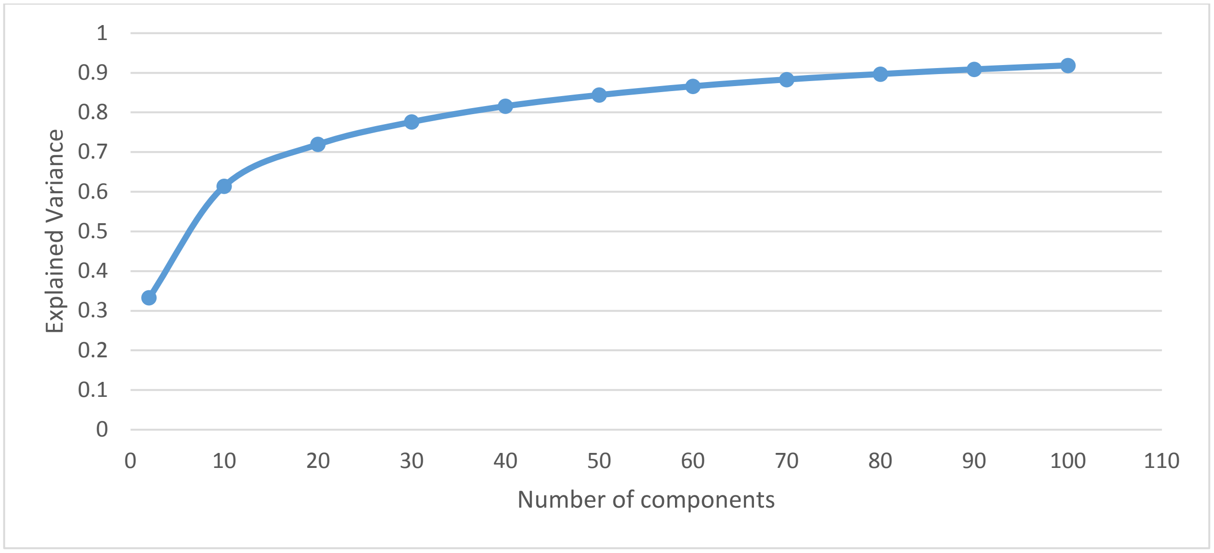
The above word-document space makes a word vector composed of 593 dimensions, with each dimension representing the word's frequency in a document. Here we try to reduce dimensions of word representation by using a dimensionality reduction approach called truncated SVD.

Truncated SVD (Truncated singular value decomposition) is a method for performing singular vector decomposition on matrix and projecting the matrix to a more compact space [1]. The resulted dimensions, also called principal components, best approximate the original distance of the matrix. It only computes the first k components as specified by user. It is a similar approach to principal component analysis, and as a matter of fact, it returns same results as PCA if we center the input matrix [1]. I chose to use this method because it works well with large sparse matrix. Also since for PCA we usually center the matrix, but here we have a sparse matrix with many 0s, making centering the data less sensible because that will convert all the zero entries to non-zeros and thus destroy the sparsity. And this is another reason I use truncated SVD because it works directly on the matrix.

I used the TruncatedSVD() method in Python's sklearn.decomposition package and first set the number of components to 20 (k=20). This TruncatedSVD projects documents from space of words to space of 20 principal components, and also it returns a linear combination of principal components for each word for performing the projection for documents between the two spaces. Then I also tried other k values as listed below.

Performance: When k=20, the explained variance from the truncated SVD model is 0.7193. Below lists and plots explained variance when k=2, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100 respectively.

K (num of components)	Explained variance
2	0.3332
10	0.6142
20	0.7193
30	0.7759
40	0.8156
50	0.8437
60	0.8658
70	0.8829
80	0.8966
90	0.9084
100	0.9184



The observation from the above chart and table is that when k (number of components) increases, the explained variance increases too. It has a much faster growth between k=2 and 30 than in the range of k=40 to 100. When k passes 60, the explained variance grows slowly when k increases.

The truncated SVD does not only provides us a mapping from the document-word space to the document-PrincipalComponents space, which is a more compact space. Moreover, the Python package also provides a PrincipalComponent-Word matrix. I use this matrix to represent a word by vector of principal components. Similar to the way in the document-word matrix, I extracted vector for words by extracting columns of the PrincipalComponent-Word matrix. The shaded column in the table below forms the vector of principal components for word “historv”.

	Word1	…	“historv”	…	Word204562
PC1			0.000152		
PC2			0.000134		
…					
PC20			0.000109		

Table. Obtaining vector for word “historv” (the shaded area).

Therefore I have built another space, a space composed of principal components as coordinates. I also generated vectors for all the words in this space by using truncated SVD. And we are ready to use the principal component based vectors to calculate the similarity between words in this new space.

3.2 Analyzing Word Similarity in the Spaces

In the two vector spaces, I calculate cosine similarity for word pairs (‘hlstory’,’historv’), (‘blue’,’hlue’), (‘historv’, ‘riage’), (‘hlue’,’riage’). Using (‘himselt,’himself’) pairs is because they are a pair of OCR error words and correct form; using the (‘hlstory’,’historv’) pair is because they are both OCR error words and have the same correct form ‘history’; using (‘historv’, ‘riage’) is because I picked a random word ‘riage’ and inspect its similarity with ‘historv’, and it is the similar reason for the pair (‘hlue’,’riage’). We would hope in a good vector space, pairs such as (‘hlstory’,’historv’) should have a high similarity, and not-so-related pairs such as (‘historv’, ‘riage’) have a low similarity.

I calculated word similarity between the four above word pairs in the document-word space and document-PrincipalComponent space. For the document-PrincipalComponent space, I also tried different k values (number of components). Cosine similarity results are shown as below.

Space	Word pair	Parameter	Cosine Similarity
document-word	‘hlstory’,’historv’	n/a	0.0283
document-word	‘himselt,’himself’	n/a	0
document-word	‘historv’, ‘riage’	n/a	0.0780
document-word	‘himselt’,’riage’	n/a	0.0267
document-PrincipalComponent	‘hlstory’,’historv’	K=10	0.8683
document-PrincipalComponent	‘himselt,’himself’	K=10	0.6072
document-PrincipalComponent	‘historv’, ‘riage’	K=10	0.5215
document-PrincipalComponent	‘himselt’,’riage’	K=10	-0.6478
document-PrincipalComponent	‘hlstory’,’historv’	K=20	0.3346
document-PrincipalComponent	‘himselt,’himself’	K=20	-0.1135
document-PrincipalComponent	‘historv’, ‘riage’	K=20	-0.2026
document-PrincipalComponent	‘himselt’,’riage’	K=20	-0.0425
document-PrincipalComponent	‘hlstory’,’historv’	K=60	-0.2790
document-PrincipalComponent	‘himselt,’himself’	K=60	0.0551
document-PrincipalComponent	‘historv’, ‘riage’	K=60	-0.3148
document-PrincipalComponent	‘himselt’,’riage’	K=60	-0.0637

Table. Cosine similarity of word pairs in different spaces.

In well-constructed space, the similarity between ‘hlstory” and “historv”, and between ‘himselt’ and ‘himself’ should be large; and the similarity between ‘historv’ and ‘riage’, and between ‘himselt’ and ‘riage’ should be small.

From the above table, for the document-word space, we can the similarity of the 4 pairs are all relatively small, and for ‘himselt’ and ‘himself’ it is even 0 because they did not appear in a same document. It is not informative in terms of semantic similarity as the similarities are in a similar small size.

For the document-PrincipalComponent space, when k increases from 10 to 60, the similarity becomes less informative. For example, when k=60, it makes ‘hlstory’ and ‘historv’ have similarity of -0.2790, a negative value, but actually we are hoping a large positive value. It seems that when k is small, e.g. k=10 here, the similarity of (‘hlstory’,’historv’), (‘himselt,’himself’), (‘himselt’,’riage’) are reasonable, as the first two are large enough and the last one is small enough. However, the similarity of (‘himselt’,’riage’) is very low, -0.6478, which does not align with our expectation.

The reason that similarity is less sensible when k is larger (e.g. k=60) can be because there are too many principal components, and although they account well for the variance, the many dimensions bring noise to the data and therefore cosine similarity is not informative in terms of the real similarity between two words. When k is smaller (e.g. k=10), the similarity seems to be reasonable, however, since there are much fewer dimensions, it may make distant words closer because of lack of dimensions. Therefore, for actual selection of best number of principal components, we need to balance between the explained variance and the meaningfulness of calculated cosine similarity.

4. K-Means Clustering on Document-Word Matrix

On top of the document-word matrix, I performed clustering on the documents using K-means clustering. The package I used is the sklearn.cluster package from Python. The purpose is to see what number of clusters can best group the 593 Darwin related documents.

I used the “Kmeans++” initialization method and set max number of iterations to 100. I tried several different number of clusters: k=10, 20, 30, 40, 50, 60, 70, 80, 90, 100. The evaluation criterion was Silhouette Coefficient, which compares the mean intra-cluster distance and the mean nearest-cluster distance. Below shows results from the clustering work. If a clustering model well groups similar instances in the same cluster then it has a higher Silhouette coefficient, and on the contrary, if it groups dissimilar objects, then the coefficient will be lower.

Num of clusters	Silhouette Coefficient
10	0.5403
20	0.5128
30	0.4045
40	0.3138
50	0.2021
60	0.2376
70	0.2943
80	0.2804
90	0.2798
100	0.2860

The above take shows that when k increases, the Silhouette coefficient has a tendency of decreasing, though not strictly decreasing. When k=50, we have the lowest Silhouette coefficient; and when k=10 we have the highest Silhouette coefficient. It looks like here we had better have a smaller k value (number of clusters), such as k=10 or k=20, to reach a good Silhouette coefficient value.

5. CMDS on Topic Modeling Data

*This part work is performed using R.

The topic modeling data provides information of topic distribution given a document. The data is probability information, for example, for document A, it has probability x of containing topic 1, and probability y of containing topic 2, and so on. The topic files can be considered as word-topic matrices.

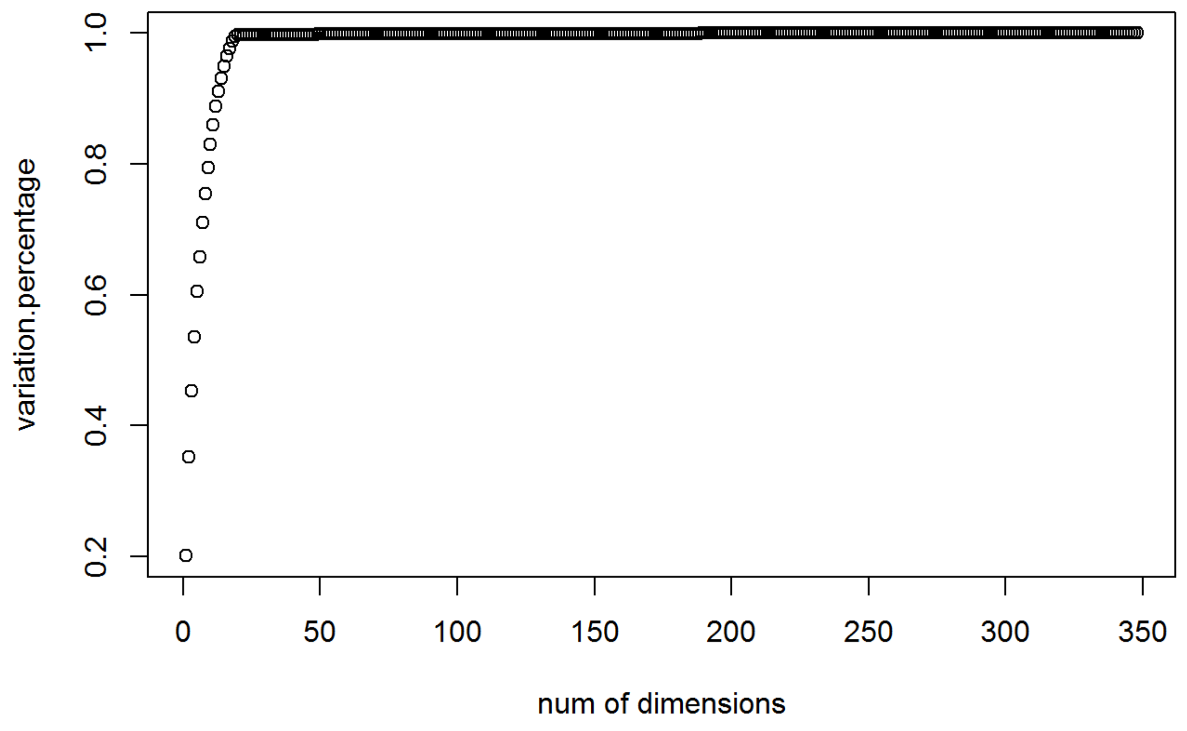
Because the data is about topic probability values, summing to 1 for each document, meaning the values are different from the normal measurement such as heights or length, and thus I chose not to apply PCA or clustering models directly on this probability data. Instead, I first computed dissimilarity matrix based on the word-topic matrices using the arccosine dissimiliarty and then apply CMDS (classical multidimensional scaling) to approximate the features of the documents.

Another source of dissimilarity comes from the KL divergence files provided by Jaimie Murdock and Colin Allen. KL divergence is computed based on document-topic matrix and estimates distance between documents based on their probability distributions. Therefore the KL divergence is a suitable input for the CMDS model too.

For both CMDS on arccosine and KL divergence, I used the topic data with number_of_topics = 20.

5.1 CMDS from arccosine

Starting from document-topic probability matrix, I computed squared roots of the matrix elements, using sum of squared root for each document to generate a cosine similarity matrix, then obtaining the arccosine value of the cosine similarity matrix, and this becomes the dissimilarity matrix as input for the CMDS model. Below plots number of dimensions in the model and the corresponding explained variance. The number of positive eigenvalues is 348 and thus the largest possible number of dimensions here is 348.



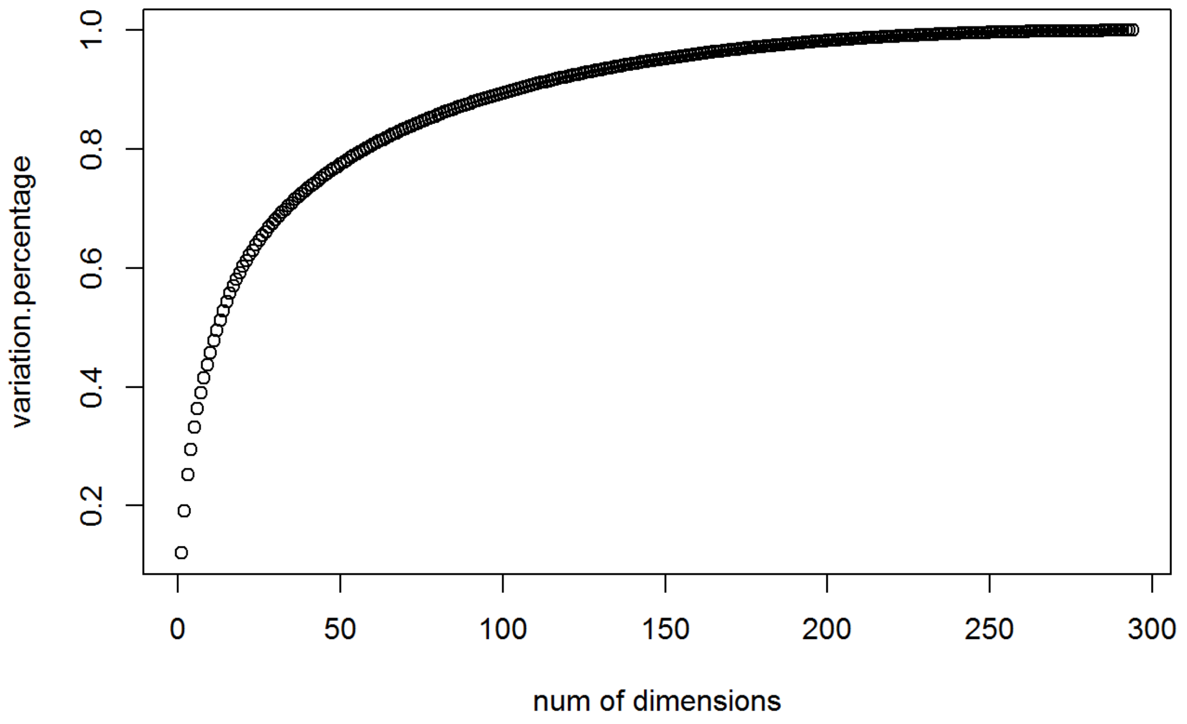
The above figure shows that the variance percentage increases as the number of dimensions (d) increases, not surprisingly. When d>18, the growth becomes very slow, as variance is already 0.9876 when d=18. Therefore it looks like here we can make the best d to d=20, as this is also the number of topics in the original topic data.

Below lists the variance values for d=1 to d=24.

```
##  [1] 0.2012382 0.3517872 0.4525244 0.5362458 0.6047942 0.6585692 0.7104484
##  [8] 0.7545336 0.7943661 0.8295818 0.8604626 0.8883077 0.9110649 0.9309931
## [15] 0.9489915 0.9644394 0.9770493 0.9876344 0.9946979 0.9970694 0.9971830
## [22] 0.9972706 0.9973494 0.9974192
```

5.2 CMDS from KL divergence

Taking advantage of the KL divergence information between documents, I used it as input for the CMDS model. KL divergence files record document to document dissimilarity based on their topic distribution. Since KL divergence is already dissimilarity, I did not perform further transformation on it. Below shows the number of dimensions (d) and the corresponding variation.



The number of positive eigenvalues is 294. The shape of the variation plot is curvier than in the arccosine case. It grows slower than the arccosine case, and it exceeds above 0.90 when d=220. However, for the arccosine case, variation has already exceeded 0.90 when d=19. Here to reach a good variation, e.g. above 0.90, we need to make d=220 or above. But this is not ideal because too many dimensions bring in noise too. To balance the variation and number of dimensions, I choose to use d=44 as when d=44 the variation is above 0.75, which is a reasonable variation.

Summarizing the arccosine and KL divergence case, using arccosine will lead to the selection of a much smaller d value, and therefore we prefer the arccosine similarity as input to the CMDS here.

References

[1] Truncated SVD in Python scikit documentation.
http://scikit-learn.org/stable/modules/decomposition.html
[2] KMeans clustering documentation in Python’s sclearn.clustering package.
http://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html#sklearn.cluster.KMeans

Python and R Code

IPython Notebook code output for this work can be downloaded at
https://iu.box.com/MiaoChenS675IIpythonCode

R Markdown code and output for this work can be downloaded at
https://iu.box.com/MiaoChenS675RCode