Latent Sentiment Analysis of Persian Text

Using Scala and Python

Bashir Sadat
Computer Science
Lehigh University
Bethlehem, PA
Sas617@lehigh.edu

ABSTRACT

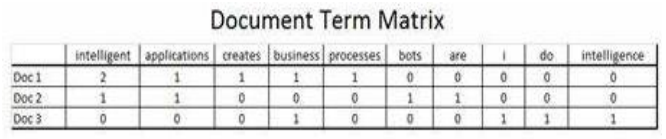
This paper presents how to use latent semantic analysis to find the relations between terms and documents of Persian text. I used both Scala and Python for doing this task. The base of the work in this paper is chapter six of the Advanced Analytics with Spark book. First, I used the latent semantic analysis to find relations between documents and terms. LSA is an unsupervised learning technique that extracts the essential meaning or most important aspects of a corpus. I used a Dataset of Persian text from Hamshahri Newspaper. However, the NLP parts of the chapter did not work on the Persian text. Therefore, I used python to process Preprocess the texts before running the LSA model on it. I explore the complications that the Persian text processing pipeline might have and propose solutions for each of the problems faced.

1 Introduction

Standard search indexes can find a given set of words in a collection of documents, but they cannot find the concepts surrounding a particular word. The latent semantic analysis helps us find those concepts even if they might not be present in the exact document. LSA is an unsupervised learning technique that extracts the essential meaning or most important aspects of a corpus into relevant concepts. We must have a clear understanding of the concept before going ahead. A concept contains threads of variation in the text and is like the subject a corpus contains. A concept has three attributes. The first is a level of relatedness, often called affinity in literature, for each document. The second is a level of relatedness for each term, and the third, is an importance score reflecting how useful the concept is in describing variance in the dataset.

LSA only works with the important concepts and ignores the concepts which have lower scores. That is why it can be a better representation for all of the corpus. LSA provides scores of similarities between terms and other terms, between documents and other documents between terms and documents. The scores are based on deeper understanding of the corpus, not just the occurrence of the words in the corpus. That is the reason behind calling it latent (hidden) semantic analysis because it does not just go by surface relationships, but finds the words and documents that have meaningful relations. The similarity measures are very useful for finding related terms and documents, allowing us to conduct different queries, which I will demonstrate later.

To get a low dimensional representation of corpora, LSA uses singular value decomposition which is a mathematical tool. The first step in the SVD calculation is maintaining a documents-term matrix. The document-term matrix contains word frequencies for each document. In other words, the matrix has the count of the number of times a term is repeated in a document.



Document Term Matrix

	intelligent	applications	creates	business	processes	bots	are	i	do	intelligence
Doc 1	2	1	1	1	1	0	0	0	0	0
Doc 2	1	1	0	0	0	1	1	0	0	0
Doc 3	0	0	0	1	0	0	0	1	1	1

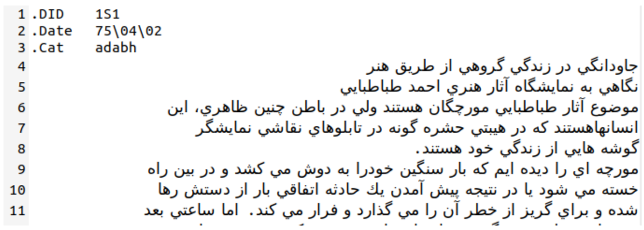
Figure 1 Document-term Matrix

For instance, the word intelligent is mentioned two times in doc1 and is not mentioned in doc3. The next step is to factorize this document-term matrix into three matrices: a matrix that expresses concepts regarding documents, a matrix that contains relatedness of concepts regarding terms, and a matrix that includes the importance for each concept in the corpus. Using these matrices, we can create a low-rank approximation of the original matrix.

2 The Dataset

In this paper, we are going to use a Persian corpus to find latent meaningful relationships between documents and terms. Working with Persian text has its complications that I will explain as we go through each section. The dataset that I am using in this paper is the Hamshahri Corpus (پیکره همشهری) which is one of the more widely available Persian corpora. Before discussing more the data, it is important to have a brief overview of the Persian language. The Persian language (also known as Farsi) is one of the popular languages in the Middle East that is spoken in several countries like Afghanistan, Tajikistan, and Iran. Persian uses Arabic-like script for writing and consists of 32 (six more characters than Arabic) characters written right to left. Documents of the Hamshahri collection are news articles of the Hamshahri newspaper from the years 1996 to 2007 (AleAhmad, Amiri, Darrudi, Rahgozar, & Oroumchian). Each document is written by a different author and has diverse language usage, from daily conversations to professionally curated texts. The documents have different data from news to entertainment. The dataset is collected by data scientists at the University of Tehran and the University of Wollongong in Dubai. For centuries Persian has been the lingua franca in the Middle East, Central Asia, and Indian subcontinent.

This created the condition where the Persian language both influenced other languages and was, itself, influenced by other languages in turn. Therefore, Persian borrows many words from different languages including European languages. That makes the NLP tasks hard on Persian texts because some of the words coming from other languages still follow the source language rules and do not follow a standardized format to form words (AleAhmad, Amiri, Darrudi, Rahgozar, & Oroumchian).
Here is a snapshot of one of the documents in the Hamshahri (fellow-citizen) newspaper collection.

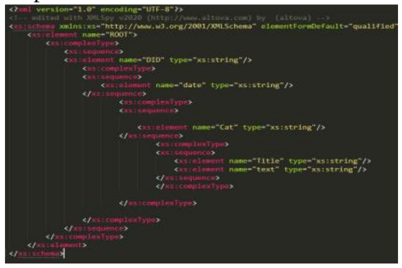


```
.DID    1S1
.Date   75\04\02
.Cat    adabh
جاودانگي در زندگي گروهي از طريق هنر
نگاهي به نمايشگاه آثار هنري احمد طباطبايي
موضوع آثار طباطبايي مورچگان هستند ولي در باطن چنين ظاهري، اين
انسانهاهستند كه در هيبتي حشره گونه در تابلوهاي نقاشي نمايشگر
گوشه هايي از زندگي خود هستند.
مورچه اي را ديده ايم كه بار سنگين خودرا به دوش مي كشد و در بين راه
خسته مي شود يا در نتيجه پيش آمدن يك حادثه اتفاقي بار از دستش رها
شده و براي گريز از خطر آن را مي گذارد و فرار مي كند.  اما ساعتي بعد
```

Figure 2 Hamshahri Collection Sample

3 Data preprocessing and LSA

First, we will divide the full document into document and title dataframe, a dataframe that has titles in one column and document text in the other.
Initially I tried to change the full text to a big XML file, the same as the chapter six of AAS. After a lot of exploration on how to change the text to XML file, I found Altova Mapforce FlexText software. To be able to use this application I needed to create an XML schema file that has the structure of the documents and titles in the same manner as a page and title schema that we can get from Wikipedia.



```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSpy v2020 (http://www.altova.com) by  (altova) -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
    <xs:element name="ROOT">
        <xs:complexType>
            <xs:sequence>
            <xs:element name="DID" type="xs:string"/>
                <xs:complexType>
                <xs:sequence>
                    <xs:element name="date" type="xs:string"/>
                </xs:sequence>
                    <xs:complexType>
                    <xs:sequence>

                        <xs:element name="Cat" type="xs:string"/>
                    </xs:sequence>
                        <xs:complexType>
                        <xs:sequence>
                            <xs:element name="Title" type="xs:string"/>
                            <xs:element name="text" type="xs:string"/>
                        </xs:sequence>
                        </xs:complexType>

                    </xs:complexType>

                </xs:complexType>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

Figure 3 XML Schema

I then had to identify each element of the schema inside the document and match them with the schema file to create the XML file.

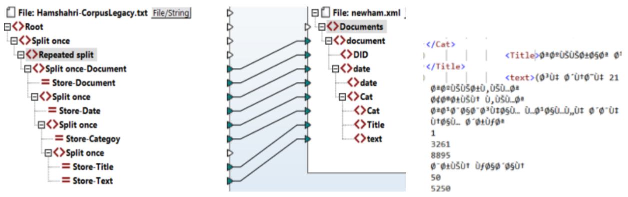
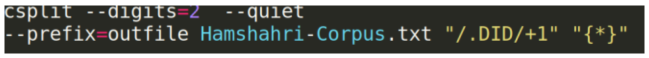


Figure 4 Altova FlexText Layout and the results.

Everything worked fine to this point and I was able to create the XML file. Given the size of the text file, it took very long, but in the end, it did not work because the software was not able to encode Persian text.

To overcome this problem, I had to use another technique. I, therefore, used the shell script to divide the documents file into files each containing one document.

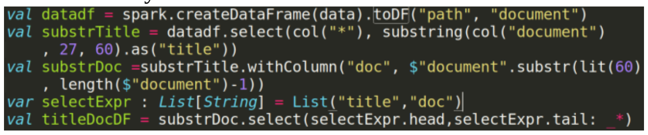


```
csplit --digits=2  --quiet
--prefix=outfile Hamshahri-Corpus.txt "/.DID/+1" "{*}"
```

Figure 5 csplit

The code snippet above divides the documents into different files, each starting with.DID.

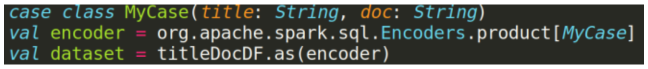
The next step is to read all the files into a Spark RDD. The first column is the path of the file and the second column is the document text. We can ignore the first column and perform all the manipulation on the document column. The index of the first letter of actual text is 27. We use this index to substring the document to create the title column as follows. The title column from 27 to 60 is almost the first line in the file. Then the rest of the document which is from index 60 to the end of the document will be as an entry of the document column.



```
val datadf = spark.createDataFrame(data).toDF("path", "document")
val substrTitle = datadf.select(col("*"), substring(col("document")
    , 27, 60).as("title"))
val substrDoc =substrTitle.withColumn("doc", $"document".substr(lit(60)
    , length($"document")-1))
var selectExpr : List[String] = List("title","doc")
val titleDocDF = substrDoc.select(selectExpr.head,selectExpr.tail: _*)
```

Figure 6 Creating Document-Title Dataframe

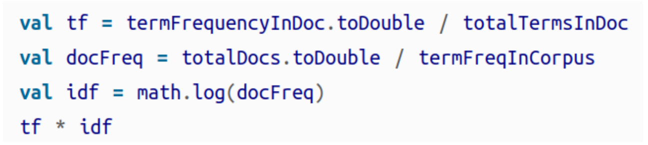
There is a small problem here, that for creating TF-IDF we need to have the data in the dataset format. For changing the data frame to the dataset we use a case class and an encoder.



```
case class MyCase(title: String, doc: String)
val encoder = org.apache.spark.sql.Encoders.product[MyCase]
val dataset = titleDocDF.as(encoder)
```

Figure 7 Dataset Encoder case class

Creating the document-term matrix is the next step in our process of LSA. As we can see in Figure1, each column represents a term in the entire corpus and each row represents a document. Then we use the TF-IDF weighting mechanism to create an importance matrix of terms and documents. There are a few assumptions that the LSA model holds. First, each document is a bag of words. This means that the sentence structure and adjacency to other words will not have any impact. Second, it just takes one copy of each word, so it is not able to figure out the meaning of the same word in a different context.



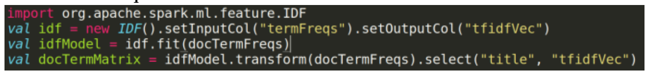
```
val tf = termFrequencyInDoc.toDouble / totalTermsInDoc
val docFreq = totalDocs.toDouble / termFreqInCorpus
val idf = math.log(docFreq)
tf * idf
```

Figure 8 TF-IDF Calculations

1. Lemmatization:
   To create the TF-IDF we need lemmatized words. There are two steps involved in this. First, we will remove stop words from the text, and then we will lemmatize the words. Removing stop words helps because they are repeated words that are not very

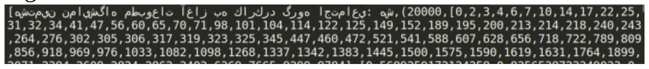
related to any document, and take up space. After this removal, we lemmatize the words. Lemmatization is the process of getting the root of a word; that is, the true meaning of a word. For instance, in English, we have Drew -> draw, ate -> eat. We use Sanford core NLP for lemmatization. While it does not produce any error with our dataset, Stanford core NLP does not work with Persian texts. However, the Stanford NLP can take a dataset of text along with the stopwords list, lemmatize the text and remove stopwords from it. I used Persian_stopwords.txt.

2. TF-IDF:
To recap, we have a dataset of sequences of terms where each sequence is a document. Going forward, we need to compute the frequency of each term in each document and the frequency of each term in the entire corpus. The spark.ml package can help us compute TF-IDF, which we can use the spark.ml package. It will do for use the calculations which is in figure 8 without the need to write function. The CountVectorizer is an Estimator that can help compute the term frequencies for us. Then we can use that to create the term frequency vector of each document. This vector has a value for each term in the document, and the value of it is the number of times that term exists in that document. We can also specify vocabulary size by removing less frequent words that can enhance the performance and reduce the noise. Keep in mind that if your dataset is small you might not reach as many words so, if that is the case, it is better to set any limit for vocabulary size. In our case we just kept the 20000 most frequent words. Now, that we have document frequency, we can calculate the inverse document frequencies. IDF is a spark estimator which uses the number of times each term appears in the corpus to inverse document frequencies.



```
import org.apache.spark.ml.feature.IDF
val idf = new IDF().setInputCol("termFreqs").setOutputCol("tfidfVec")
val idfModel = idf.fit(docTermFreqs)
val docTermMatrix = idfModel.transform(docTermFreqs).select("title", "tfidfVec")
```

Figure 9 Calculate TF-IDF



```
[هشتمین نمایشگاه مطبوعات آغاز به کارکرد گروه اجتماعی: شهر,(20000,[0,2,3,4,6,7,10,14,17,22,25,
31,32,34,41,47,56,60,65,70,71,98,101,104,114,122,125,149,152,189,195,200,213,214,218,240,243
,264,276,302,305,306,317,319,323,325,345,447,460,472,521,541,588,607,628,656,718,722,789,809
,856,918,969,976,1033,1082,1098,1268,1337,1342,1383,1445,1500,1575,1590,1619,1631,1764,1899,
```

Figure 10 Document-Term Matrix, some of first entry

Noticed that everything is changed to numbers. If we want to know what term a number is referring to, we need a mapping of the vocabulary and their positions. In this way we can trace what those numbers are referring to.

3. Singular Value Decomposition:
Now that we have the base matrix for our SVD calculations, the docTermMatrix calculated earlier (figure 9) will be the base of our work. We will call this matrix M going forward. Spark MLlib offers an implementation of SVD that takes matrix M(m x n) and gives three factors of it as matrices. If we multiply all of the three factored matrices, we will be very near to the M.

	d_1	d_2	d_3	d_4	d_5	d_6
ship	1	0	1	0	0	0
boat	0	1	0	0	0	0
ocean	1	1	0	0	0	0
voyage	1	0	0	1	1	0
trip	0	0	0	1	0	1

Figure 11 docTermMatrix or M matrix, also called the original matrix.

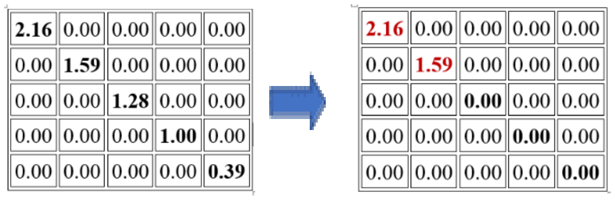
$M \approx U\,S\,V^T$

m= number of documents.

n= number of terms

k = less than or equal to n denotes the number of concepts that we want to keep.

- U is an m × k matrix, which is called documents space.
- S was sometimes written as ∑ is a k × k that shows the strength of each concept in the corpus. K=2, in this case.



2.16	0.00	0.00	0.00	0.00
0.00	**1.59**	0.00	0.00	0.00
0.00	0.00	**1.28**	0.00	0.00
0.00	0.00	0.00	**1.00**	0.00
0.00	0.00	0.00	0.00	**0.39**

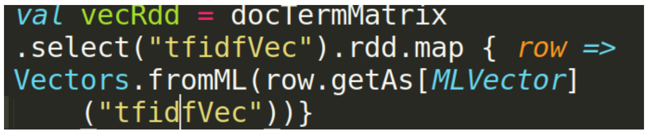
2.16	0.00	0.00	0.00	0.00
0.00	**1.59**	0.00	0.00	0.00
0.00	0.00	**0.00**	0.00	0.00
0.00	0.00	0.00	**0.00**	0.00
0.00	0.00	0.00	0.00	**0.00**

- Finally, we have V^T also called term space.

$$\underset{U}{\begin{bmatrix} .13 & .02 & -.01 \\ .41 & .07 & -.03 \\ .55 & .09 & -.04 \\ .68 & .11 & -.05 \\ .15 & -.59 & .65 \\ .07 & -.73 & -.67 \\ .07 & -.29 & .32 \end{bmatrix}} \underset{\Sigma}{\begin{bmatrix} 12.4 & 0 & 0 \\ 0 & 9.5 & 0 \\ 0 & 0 & 1.3 \end{bmatrix}} \underset{V^T}{\begin{bmatrix} .56 & .59 & .56 & .09 & .09 \\ .12 & -.02 & .12 & -.69 & -.69 \\ .40 & -.80 & .40 & .09 & .09 \end{bmatrix}}$$

Figure 12 SVD, (Ethen, 2020)

Spark MLlib has an implementation of SVD, but it operates on RDD. Therefore, we need our data frame to have an RDD format. The following code snippet converts it to RDD.



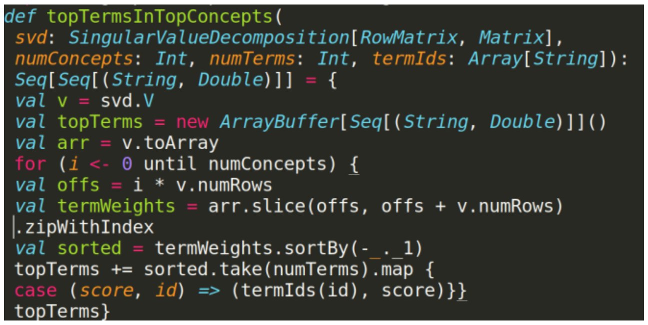
```
val vecRdd = docTermMatrix
.select("tfidfVec").rdd.map { row =>
Vectors.fromML(row.getAs[MLVector]
    ("tfidfVec"))}
```

Figure 13 Converting Dataframe to RDD

Important concepts

The V matrix represents concepts through the terms that are important to them. It contains a column for every concept and a row for every term. The value at each position means the relevance of that term to that concept. Here are the concepts documents and terms.

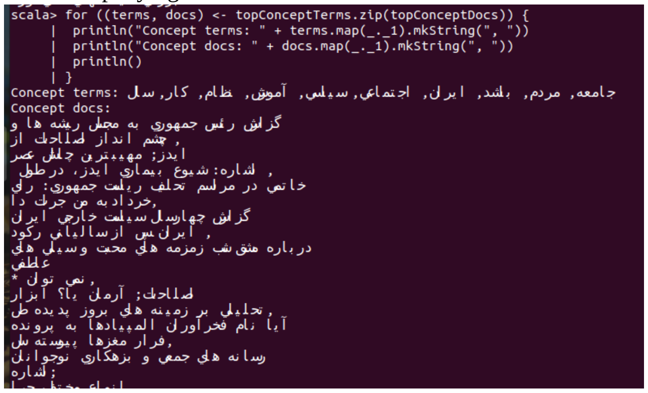
The code snippet below can find us the top concepts and most relevant terms to them.



```
def topTermsInTopConcepts(
 svd: SingularValueDecomposition[RowMatrix, Matrix],
 numConcepts: Int, numTerms: Int, termIds: Array[String]):
 Seq[Seq[(String, Double)]] = {
 val v = svd.V
 val topTerms = new ArrayBuffer[Seq[(String, Double)]]()
 val arr = v.toArray
 for (i <- 0 until numConcepts) {
 val offs = i * v.numRows
 val termWeights = arr.slice(offs, offs + v.numRows)
 .zipWithIndex
 val sorted = termWeights.sortBy(-_._1)
 topTerms += sorted.take(numTerms).map {
 case (score, id) => (termIds(id), score)}}
 topTerms}
```

Figure 14 Top terms in top concepts code

Additionally, we can get the documents related to the top concepts in the same way. In the picture below we can see the results which is surprisingly good, even though we have not done any Persian NLP task on like lemmatization and stemming. Just by tokenizing and removing stopwords we got quite good results. Please refer to the accompanying codes for the rest of the code on this.



```
scala> for ((terms, docs) <- topConceptTerms.zip(topConceptDocs)) {
     |  println("Concept terms: " + terms.map(_._1).mkString(", "))
     |  println("Concept docs: " + docs.map(_._1).mkString(", "))
     |  println()
     | }
Concept terms: جامعه, مردم, بلشد, ایرلن, اجتماعي, سيلسي, آموش, ظام, كار, سلل
Concept docs:
گزلش رئيس جمهوري به مجلس ريشه ها و
چشم انداز صلاحلت از ,
ايدز; مهيبترين چللش عصر
لشاره:شيوع بيماري ايدز، درطول ,
خاتمي در مرلسم تحلف ريلست جمهوري: رلي
خرداد به من جرلت دا,
گزلش چهارسلل سيلست خارجي ايرلن
ايرلن بس ازساليلني ركود ,
درباره مثق شب زمزمه هلي محبت وسيلي هلي
علطفي
* نمي تولن ,
صلاحلت; آرملن يا؟ ابزار
تحليلي بر زمينه هلي بروز پديده ط,
آيا نام فخرآورلن المپيادها به پرونده
فرار مغزها پيوسته لش,
رسانه هلي جمعي و بزهكاري نوجوانلن
لشاره;
```

Figure 15 Top terms in top concepts

Now is the time to further explore the relations between terms and terms, terms and documents, and relatedness of a group of words with documents. If we had used the original matrix for finding documents and terms relevance, we could use cosine similarity scores. Cosine similarity is based on the angles between two vectors.

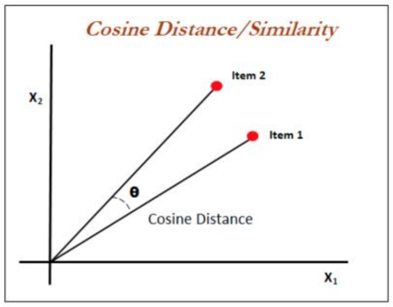


Figure 16 cosine similarity (Dangeti, 2017)

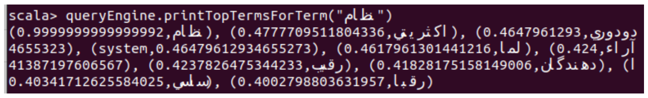
For instance, if we want to find the relevance between two documents, we can find the cosine similarity between their row vectors. But this similarity score and relevance is not from the latent factors but just a shallow knowledge which is just the frequency counts. LSA goes deeper than that. An example from our dataset, if the word ("جامعه", society) is not available in the document " آموزش وپرورش عالي. " (Higher Education) but it has many occurrences of the word "نظام" (government system), there is a high chance that LSA finds the relation between the document and ("جامعه", society) based on its relation with the word "نظام" (government system).

Based on this, it is now possible to use the work we have done so far to query from the results of LSA. We use the LSAQueryEngine class developed by the authors of the AAS book (Ryza, Wills, Laserson, & Owen, 2017).

Term-Term Relevance (Unknown, 2020)

LSA finds the relations between two terms by calculating the cosine similarity between their two columns in the low-ranked matrix. Low ranked matrix is the matrix that will be generated if we multiply the three matrices back together after reducing their dimensions. It offers a better representation of data in the following ways. Taking care of synonyms by condensing terms near each other. And by placing fewer weights on the word which has several meanings it accounts for polysemy. Finally, it will get rid of noise taking the most important concepts.

But there is even a better way. Cosine similarity between two columns in the low-ranked matrix is equal to the cosine similarity between that two columns in SV^T. This is the same for finding similarity between a term and a sequence of terms. We can use SV for it.



```
scala> queryEngine.printTopTermsForTerm("ظام")
(0.9999999999999992,ظام), (0.4777709511804336,اكتريقي), (0.4647961293,دودوري
4655323), (system,0.46479612934655273), (0.4617961301441216,لما), (0.424,آراء
41387197606567), (0.4237826475344233,رقيب), (0.41828175158149006,دهندگلن), (ا
0.40341712625584025,سلسي), (0.4002798803631957,رقبا)
```

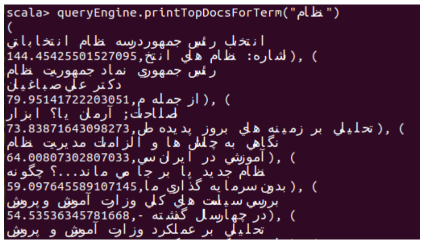
Figure 17 Term-term relevance

There is an interesting result in the output in figure 17. It found a relation between the queried term ("نظام") and the English term "system". It looks like the English word system has been used along with the Persian word in many cases.

Document-Document Relevance

Similarly, to find relevance between two documents we can use cosine similarity between their corresponding rows in the US, we do not need to use the low-ranked matrix, which is bigger than the US, S is also called sigma in some sources like in figure 12.

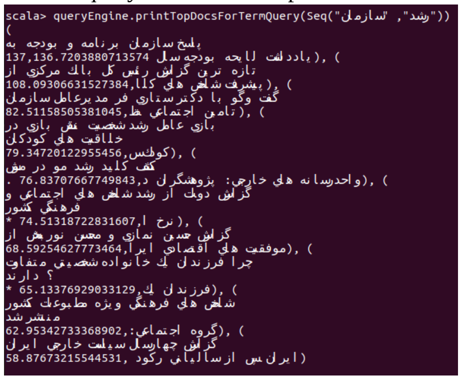
In the same way can calculate the document term relevance which is the u_d^T S v_t.



```
scala> queryEngine.printTopDocsForTerm("ظام")
(
انتخلب رئيس جمهوردرسه ظام انتخاباتي
144.45425501527095,انتخ ظام هلي :لشاره), (
رئيس جمهوري نماد جمهوريت ظام
دكتر علي صباغيلن
79.95141722203051,از جمله م), (
صلاحلت; آرملن يا؟ ابزار
73.83871643098273,تحليلي بر زمينه هلي بروز پديده ط), (
نگاهي به چللش ها و الزامك مديريت ظام
64.00807302807033,آموزشي در ايرلن سي), (
ظام جديد پا برجا مي ماند...؟ چگونه
59.097645589107145,بدون سرمايه گذاري ما), (
بررسي سيلست هلي كلي وزارت آموش وپروش
54.53536345781668,در چهارسلل گذشته -), (
تحليلي بر عملكرد وزارت آموش و پروش
```

Figure 18 top documents for a term

We can also have query to search multiple terms relations.



```
scala> queryEngine.printTopDocsForTermQuery(Seq("سازمان" ,"رشد"))
(
پاسخ سازمان برنامه و بودجه به
137,136.7203880713574 یادداشت لایحه بودجه سال), (
تازه ترین گزارش رئیس کل بانک مرکزی از
108.09306631527384,پیشرفت شاخص های کلان), (
گفت وگو با دکتر ستاری فر مدیرعامل سازمان
82.51158505381045,تامین اجتماعی ط), (
بازی عامل رشد شخصیت نقش بازی در
خلاقیت های کودکان
79.34720122955456,کودکس), (
کشف کلید رشد مو در موش
. 76.83707667749843,واحدرسانه های خارجی: پژوهشگران د), (
گزارش دولت از رشد شاخص های اجتماعی و
فرهنگی کشور
* 74.51318722831607,نرخ ا), (
گزارش حسین نمازی و محسن نوربخش از
68.59254627773464,موفقیت های اقتصادی ایرا), (
چرا فرزندان یک خانواده شخصیتی متفاوت
دارند ؟
* 65.13376929033129,فرزندان یک), (
شاخص های فرهنگی ویژه مطبوعات کشور
منتشر شد
62.95342733368902,گروه اجتماعی:), (
گزارش چهارسال سیاست خارجی ایران
58.87673215544531, ایران پس ازسالیانی رکود)
```

Figure 19 top documents for term query

The results look satisfactory for the task that we were planning to do. But there is one problem that the lemmatization has not worked properly in our text, so we did not get to the root of the words. To address the problem, I tried to find NLP tools in Scala, but I was not able to do that, so I switched to python and done the rest of the work on python using different NLP tools, more on section 4.

4 Persian Natural Lagunage Processing in Python

As I discussed before, Persian text has it is complications and not many tools available for it. Therefore, I had to switch to Python to do further NLP experiments on my dataset, as I could not find any tool for processing Persian text in Spark. In this paper I used three NLP packages in python.

The first one is famous NLP tool NLKT which stands for Natural Language Toolkit. I used NLTK for stop words removal purpose because it is simple and by adding stop words file into certain directory it can remove all the words that you do not want in your data. For instance, one word that was not removed when creating documents title dataframe in spark is .DID, I added this word into the stop words file and NLTK was able to delete it.

Another package that I used here was Parsivar package for python. Parsivar is a Persian text preprocessing tool that can be used for text normalization, half-space correction, word, and sentence tokenization (splitting words and sentences) word stemming and a few more. I used this package mainly because of it is half-space removal capacity. Half-space is a Persian language trait that allows two words to be one word while have smaller space between them. In a normal NLTK or any other NLP tool they will be considered as two words, but the Parsivar will remove the extra space and make sure they are read as one word.

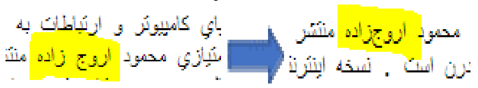


Figure 20 Half-space removal

We can see that word Oroj-Zadeh is written separately while it is one word, someone might write that as Orojzadeh if it was English. Another good thing about the Parsivar package is that it has spellcheck as well. Since we substring from the document and title entries we might have broken some words in the process, the package will help us to remove or correct those words.

The third package that I used is called Hazm package, the word Hazm means digest in Persian, which is good naming. I initially started with this package if it was not for the Half-space problem this package has all the other tools needed for processing the Persian text.

4.1 NLP pipeline for Hamshahri collection

There are many steps in removing the extraneous strings and characters from the text. For instance, we want to remove any digit information in our text or remove hidden characters in the text that can be interpreted as a word and affect the result of our work. The figure below shows how the pipeline is ordered. The output of one process is fed as input to the next of it.

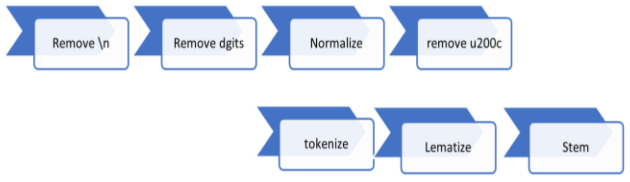
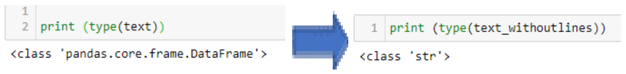


Figure 21 NLP Pipeline

4.1.1 To be able to have the text ready to work in the normalization process we needed to remove the "\n" markers from the text, otherwise, we will be faced with an error. The error says that it expects a string object but having the line number indicators turns our text into a data frame. To solve it we will remove the line indicators.



```
print (type(text))
<class 'pandas.core.frame.DataFrame'>
```

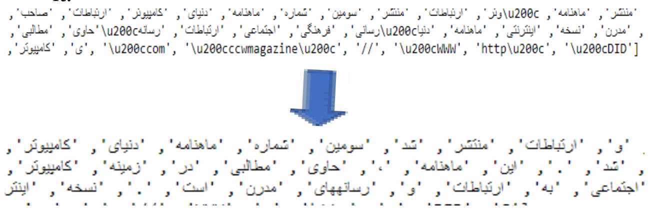
```
print (type(text_withoutlines))
<class 'str'>
```

Figure 22 Line-indicator Removal

4.1.2 One more important point is needed to be done before normalization, we need to remove digits from our text. Because the normalizer changes the digits into Persian digits and will be considered words by the LSA if we do not remove them now. We do not need any complicated technique for that we will just replace them with "".

4.1.3 Then we normalize the text to remove virtual spaces, extra characters, and other unnecessary strings. [more on this]

4.1.4 The next step is tokenizing the text into words. To do so we will feed our normalized text into the word_tokenizer of the Parsivar package. It works well but there is a problem that we have to address. We can see that there is u200c added to some of our words. It is because when normalized it added u200c instead of the half-spaces. We do not want that so we can remove those characters. Note that there is a sentence tokenizer tool as well but for our purpose, we will not need it.

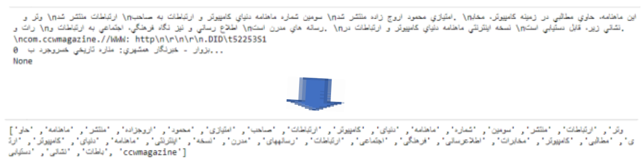


```
'منتشر', 'ماهنامه', 'u200c\وتر', 'ارتباطات', 'منتشر', 'سومین', 'شماره', 'ماهنامه', 'دنیای', 'کامپیوتر', 'ارتباطات', 'صاحب',
, 'مدرن', 'نسخه', 'اینترنتی', 'ماهنامه', 'دنیاu200c\رسانی', 'فرهنگی', 'اجتماعی', 'ارتباطات', 'رسانهu200c\'حاوی', 'مطالبی',
,'کامپیوتر' ,'ی', '\u200ccom', '\u200ccwmagazine\u200c', '//', '\u200cWWW', 'http\u200c', '\u200cDID']
```

```
'و', 'ارتباطات', 'منتشر', 'شد', 'سومین', 'شماره', 'ماهنامه', 'دنیای', 'کامپیوتر',
, 'شد', '.', 'این', 'ماهنامه', '،', 'حاوی', 'مطالبی', 'در', 'زمینه', 'کامپیوتر',
'اجتماعی', 'به', 'ارتباطات', 'و', 'رسانههای', 'مدرن', 'است', '.', 'نسخه', 'اینتر
```

Figure 23 u200c problem solution

4.1.5 Having the tokenized words in hand we will proceed to remove stop words. We will use the NLTK package to remove those words. Here I wandered a lot to figure out how to do the stop word removal far Farsi words. Both of the Persian NLP packages do not have a tool for removing stop words. I found a Persian stop words dataset online and then put it into the NLTK stop words directory (nltk_data\corpora\stopwords). This way it detects stop words file and remove them from the text. Another important work that this file does is the removal of URLs and .DID or any other unwanted string or character. In the tokenization step we divided URLs into tokens, now we can put the “ com, http, URL” or other repetitive words in our stop words file and NLTK will remove them from the text. Still there will be some unwanted words, but the good news is that they are not as many of them that enables the low rank approximation technique in LSA to discard them later.

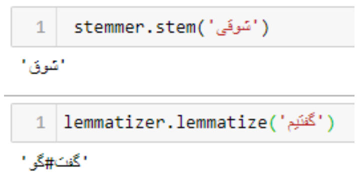


```
این ماهنامه، حاوی مطالبی در زمینه کامپیوتر. مخابn\ .امتیازی محمود اروج زاده منتشر شدn\ سومین شماره ماهنامه دنیای کامپیوتر و ارتباطات به صاحبn\ ارتباطات منتشر شدn\ وتر و
.نشانی زیر. قابل دستیابی استn\ نسخه اینترنتی ماهنامه دنیای کامپیوتر و ارتباطات درn\ .رسانه های مدرن استn\ اطلاع رسانی و نیز نگاه فرهنگی، اجتماعی به ارتباطات وn\ رات و
\ncom.ccwmagazine.//WWW: http\n\r\n\r\n.DID\t5225351
0  بزوار - خبرنگار همشهري: مناره تاريخي خسروجرد ب...
None
```

```
['حاو', 'ماهنامه', 'منتشر', 'اروجزاده', 'محمود', 'امتیازی', 'صاحب', 'ارتباطات', 'کامپیوتر', 'دنیای', 'ماهنامه', 'شماره', 'سومین', 'منتشر', 'ارتباطات', 'وتر', '
ی', 'مطالبی', 'کامپیوتر', 'مخابرات', 'اطلاعرسانی', 'فرهنگی', 'اجتماعی', 'ارتباطات', 'رسانههای', 'مدرن', 'نسخه', 'اینترنتی', 'ماهنامه', 'دنیای', 'کامپیوتر', 'ارژ
دستیابی', 'نشانی', 'باطات', 'ccwmagazine']
```

Figure 24 Stop words and unwanted characters removal.

4.1.6 Now that changed our text into tokens, now is the time to lemmatize each of the token. There are two ideas about lemmatization that worth explaining, stemming, and lemmatizing. First, stemming refers to the practice of shortening a word to make it look more closer to its original form. For instance, in English Onions ->onion, walked-> walk, it is the same thing in Persian. And lemmatization is the process of getting the root of a word, or the true meaning of word. For instance, in English, we have Drew -> draw, ate -> eat . Here is an example of both in Persian. (Khallash & Imany, 2020)



```
stemmer.stem('شوقی')
```

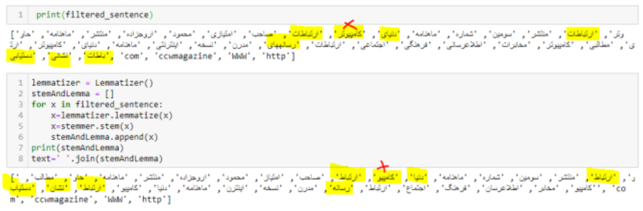
```
'شوق'
```

```
lemmatizer.lemmatize('گفتیم')
```

```
'گفت#گو'
```

Figure 25 Stemmer and Lemmatizer

The final result looks good, although there some words that did not get stemmed and lemmatized correctly. But that will do the job for our purpose.



```
print(filtered_sentence)
```

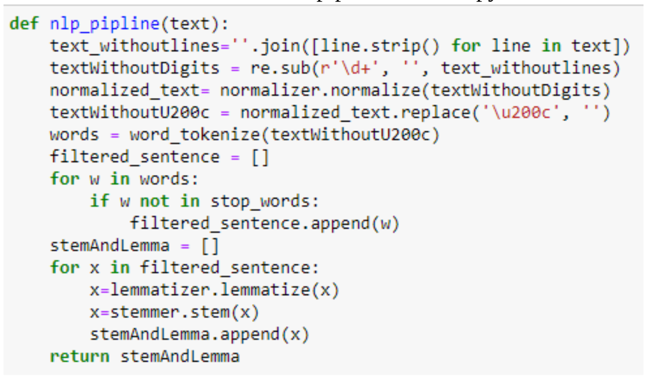
```
['حاو', 'ماهنامه', 'منتشر', 'اروجزاده', 'محمود', 'امتیازی', 'صاحب', 'ارتباطات', 'کامپیوتر', 'دنیای', 'ماهنامه', 'شماره', 'سومین', 'منتشر', 'ارتباطات', 'وتر', '
ی', 'مطالبی', 'کامپیوتر', 'مخابرات', 'اطلاعرسانی', 'فرهنگی', 'اجتماعی', 'ارتباطات', 'رسانههای', 'مدرن', 'نسخه', 'اینترنتی', 'ماهنامه', 'دنیای', 'کامپیوتر', 'ارژ
دستیابی', 'نشانی', 'باطات', 'com', 'ccwmagazine', 'WWW', 'http']
```

```
lemmatizer = Lemmatizer()
stemAndLemma = []
for x in filtered_sentence:
    x=lemmatizer.lemmatize(x)
    x=stemmer.stem(x)
    stemAndLemma.append(x)
print(stemAndLemma)
text=' '.join(stemAndLemma)
```

```
['مطالب', 'حاو', 'ماهنامه', 'منتشر', 'اروجزاده', 'محمود', 'امتیاز', 'صاحب', 'ارتباط', 'کامپیو', 'دنیا', 'ماهنامه', 'شماره', 'سومین', 'منتشر', 'ارتباط', 'ر
دستیاب', 'نشان', 'ارتباط', 'کامپیو', 'دنیا', 'ماهنامه', 'اینترن', 'نسخه', 'مدرن', 'رسانه', 'ارتباط', 'اجتماع', 'فرهنگ', 'اطلاعرسان', 'مخابر', 'کامپیو', 'co
m', 'ccwmagazine', 'WWW', 'http']
```

We can see that the one word which is not working is a computer (کامپیوتر), it is because it is ended with letters (تر) which is used for comparison, like the (بهتر) which is made of (به) and (تر). That is how Persian makes new words. But the computer is an English word and translated in Persian that should be an exception, but the package has not considered these exceptions. I might explore this later to add these exceptions in the Persian packages.

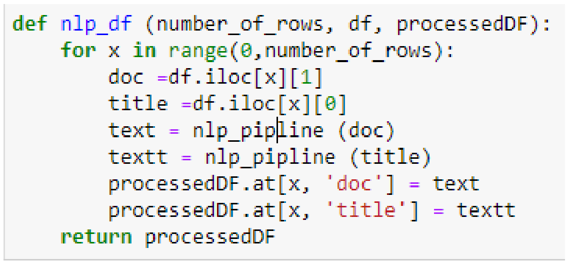
Here is the full code of the NLP pipeline code in python.



```
def nlp_pipline(text):
    text_withoutlines=''.join([line.strip() for line in text])
    textWithoutDigits = re.sub(r'\d+', '', text_withoutlines)
    normalized_text= normalizer.normalize(textWithoutDigits)
    textWithoutU200c = normalized_text.replace('\u200c', '')
    words = word_tokenize(textWithoutU200c)
    filtered_sentence = []
    for w in words:
        if w not in stop_words:
            filtered_sentence.append(w)
    stemAndLemma = []
    for x in filtered_sentence:
        x=lemmatizer.lemmatize(x)
        x=stemmer.stem(x)
        stemAndLemma.append(x)
    return stemAndLemma
```

Figure 26 NLP Pipeline Code

To apply that on all rows and columns I wrote another function to iterate over rows.



```
def nlp_df (number_of_rows, df, processedDF):
    for x in range(0,number_of_rows):
        doc =df.iloc[x][1]
        title =df.iloc[x][0]
        text = nlp_pipline (doc)
        textt = nlp_pipline (title)
        processedDF.at[x, 'doc'] = text
        processedDF.at[x, 'title'] = textt
    return processedDF
```

Figure 27 Apply on all rows of the data frame.

Now, that we have this data frame we can save it as a .csv file and import it to Spark, and continue forward from step 6 of section three.

5 Conclusion

In this work, I worked with latent semantic analysis to understand Persian text. The base of my work was the AAS book chapter 6 (Ryza, Wills, Laserson, & Owen, 2017). I was able to use the LSA model that uses the Singular Value Decomposition for creating a low-ranked approximation of the document-term matrix, to understand Persian text. The results of my work show that the LSA model gives us a result on Persian text. To improve the quality of the results, I used python to clean and process the Persian text for LSA. I used three packages in Python to clean the dataset. NLTK, Hazm, and Parsivar are the three packages that I used which complete each other.

References

AleAhmad, A., Amiri, H., Darrudi, E., Rahgozar, M., & Oroumchian, F. (n.d.). *Hamshahri: A Standard Persian Text Collection.* Electrical and Computer Engineering Department, University of Tehran , University of Wollongong in Dubai. University of Tehran.

Brunton, S. L., & Kutz, J. N. (2017). *Data Driven Science & Engineering*. University of Washington.

Dangeti, P. (2017). *Statistics for Machine Learning.* Birmingham : Packt Publishing Ltd.

Ethen. (2020). *machine-learning*. Retrieved from ethen8181.github.io: http://ethen8181.github.io/machine-learning/dim_reduct/svd.html

Khallash , M., & Imany, M. (2020). *hazm*. Retrieved from https://github.com/sobhe: https://github.com/sobhe/hazm

Manning, C. D. (2008). *Introduction to Information Retrieval.* Cambridge University Press.

Ryza, S., Wills, J., Laserson, U., & Owen, S. (2017). Understanding Wikipedia with Latest Semantic Analysis. In *Advanced Analytics with Spark, 2nd Edition.* O'Reilly Media, Inc.

Unknown. (2020). *Parsivar*. Retrieved from https://github.com/ICTRC/: https://github.com/ICTRC/Parsivar