Lab 7, NumPy and the Jokes

Due date: Monday, May 2, 11:59pm.

Overview

For this assignment you will create/edit a number of Jupyter notebooks in which you will work with `NumPy` arrays (as well as other data structures) to store, represent and analyze some data from the **Jester**[**?**] dataset.

Assignment Preparation

This is an individual assignment.

Data

You will be using *joke ratings data* from the **Jester** project, run by Professor Ken Goldberg at UC Berkeley. **Jester**[**?**] is an on-line joke recommender system available at

http://shadow.ieor.berkeley.edu/humor/

Disclaimer. Please read this note before proceeding! Jester has a database consisting of 100 jokes. The jokes are shown to the user, and the user's reaction to them is measured on a continuous scale. **Please be aware** that the jokes available through `Jester` may contain some examples that you personally will find objectionable. Please know, that *it is not my intent to offend anyone's sensibilities* (and neither is it the intent of the authors of the dataset and **Jester**).

The Dataset

The portion of the **Jester** dataset that you will be studying consists of two files uploaded to the `/home/dekhtyar/data` directory on the Jupyter server.

List of jokes. The first file is `Jokes.xml`. This file contains the list of jokes that **Jester** uses. The file has the following simple format:

```
<jokes>
  <joke>
    Joke 1 goes here...
  </joke>
   ...
  <joke>
     Joke 100 goes here...
  </joke>
</jokes>
```

There are 100 jokes in the file — these are all the jokes **Jester** uses. This file can be easily parsed with `BeautifulSoup` using the techniques you have aquired from the previous labs. The jokes are not numbered internally in the XML file, but their order corresponds to the joke ids in the ratings data file.

Ratings Data. The `jester-data-1.csv` file contains our joke ratings data. The full dataset contains three data files of roughly equal size. Of the three files, you will be using only the first one, `jester-data-1`.

The **Jester** dataset web page describes the format of the data as follows[**?**]:

> "Format:
>
> 1. 3 Data files contain anonymous ratings data from 73,421 users.
> 2. Data files are in .zip format, when unzipped, they are in Excel (.xls) format
> 3. Ratings are real values ranging from -10.00 to +10.00 (the value "99" corresponds to "null" = "not rated").
> 4. One row per user
> 5. The first column gives the number of jokes rated by that user. The next 100 columns give the ratings for jokes 01 - 100.
> 6. The sub-matrix including only columns $\{5, 7, 8, 13, 15, 16, 17, 18, 19, 20\}$ is dense. Almost all users have rated those jokes (see discussion of "universal queries" in the above paper)."

There are 24,983 rows (ratings records) in the `jester-data-1.csv` file. Each row has 101 values as described above.

Lab Assignment

For this assignment, you will build three Jupyter notebooks (some of them, using `NumPy`):

1. `JokeIndex.ipynb`: in this notebook you will build Python code for reading the jokes file, processing it, and creating an inverted index identifying which words are used in which joke.

2. `Ratings.ipynb`: in this notebook you will import the **Jester** ratings matrix and perform some manipulations and operations on it to extract some information. You will actively use `NumPy` functionality for this.

3. `Questions.ipynb`: in this notebook, you will combine the two data structures built in the other two notebooks (the inverted index and the ratings matrix) and will answer some questions regarding the users' attitudes towards specific jokes in the dataset.

JokeIndex.ipynb

Your goal for this notebook is to create an inverted index of the list of jokes. To do so:

1. Grab the `Jokes.xml` file and split it into individual jokes, represented as `string`s using `BeautifulSoup`.

2. Tokenize the text of each joke, and remove stop words. You can use the stopword list from **Lab 4** (it is available at `/home/dekhtyar/data/stopwords.txt`.

3. Create an inverted index. An inverted index associates a list of jokes that contain a specific term with the term itself. It can be represented in Python as a dictionary, where the keys are the terms and the values are lists of joke ids.

   You can use integers from 1 to 100 as ids of individual jokes. Make sure to store the lists of joke ids in sorted order - this may come in hand later.

4. Implement basic search functionality. Write the following functions that walk the inverted index (you can treat the inverted index as a global variable in your notebooks) and do the following:

   - `findJokes(word)`: returns the list of Joke ids for all jokes that contain a given word.
   - `findAllJokes(wordList`: returns the list of Joke ids for all jokes that contain **all** the words from a given list.

     Use the notebook to test the functionality of these two functions.

Ratings.ipynb

Your goal for this notebook is to create a `NumPy` array storing the information about the user ratings of the jokes, as well as a few variants of the data. Proceed as follows:

1. Read the array from the file into a `NumPy` array variable `rawRatings`. The result shall be a 2D `NumPy` array with 24,983 and 101 columns.
2. Remove the first column. Create a new `NumPy` array variable `rawRatingsTable` and put into this variable the contents of `rawRatings` without the first column. That is, `rawRatingsTable` shall contain all the user ratings, without the column specifying how many jokes each user rated.
3. Preserve the first column. Turn the first column of `rawRatings` into its own `NumPy` array, `userActivity`.
4. Dealing with null values. Create a version of the `rawRatingsTable` called `ratingsTable` which replaces the 99 values (indicating no rating from a user) with a 0.
5. Create a dense submatrix. Create a `NumPy` array variable `denseRatings` which contains user ratings for jokes $\{5, 7, 8, 13, 15, 16, 17, 18, 19, 20\}$ (as mentioned above, this is a dense matrix).
6. Create another dense submatrix. Find all users who scored all the jokes and create a `NumPy` array variable `activeUsers` that stores all their ratings.
7. Run some simple analysis. Split the `denseRatings` array into two: `denseR1` and `denseR2`. Both arrays preserve the columns. `denseR1` array contains the same rows as `activeUsers` array (users who rated all the jokes), while `denseR2` array contains all the remaining rows.
8. Compare the scores. For each of the jokes featured in `denseR1` and `denseR2` matrices, compare the average scores the joke received. Which jokes were preferred by users who rated all the jokes? Which jokes were preferred by users who rated fewer jokes?
9. Find the highest rated joke. Find the highest rated joke by all users, and the joke that was the highest rated joke for only the users who rated all the jokes.

   Note: When performing the computations, please exclude people who did not rate a joke from consideration.

Questions.ipynb

In this notebook, you will analyze the ratings data with the help of the inverted index on the jokes contents.

1. Bring over from the other notebooks all the code necessary to create the inverted index for the jokes collection, and to create the ratings matrix (whichever full ratings matrix of the ones from the `Ratings.ipynb` that you are most comfortable working with).
2. Add code that finds answers to the list of questions stated below.

Questions.

1. Do people like Bill Clinton jokes? A simple way of studying the answer to this question is to

   - Identify the terms that are likely to be present in Bill Clinton jokes.
   - Study the average score of Bill Clinton jokes
   - Compare it to the average score of other jokes.

   You can be a bit more in-depth in your studies. In **Lab 8** or **Lab 9** you will actually conduct formal statistical analysis and will be able to answer these questions more precisely. For this lab, just concentrate on getting the right data.

2. Do people prefer longer or shorter jokes? To answer this question, create a 100x2 `NumPy` array that, for each joke (rowId), stores the number of non-stop words found in it, and the average score for the joke. We will see if the regression is there later, but for this assignment, simply compare the average scores of the longest 25 jokes, with the average scores of the shortest 25 jokes.

3. Determine the most polar joke. Split all scores for each joke into positive and negative. Compute the average positive score. Compute the average negative score. The polarity score is

   $$polarity(joke) = \frac{n_{pos}}{n} \cdot |avg_{pos})| + \frac{n_{neg}}{n} \cdot |avg_{neg}|$$

   where n_{post} and n_{neg} are the numbers of positive and negative ratings, n is the total number of ratings, and avg_{pos} and avg_{neg} are the average positive and average negative scores respectively.

   Find the joke that has the highest polarity, report it.

Deliverables and submission instructions

The information about fetching the notebooks and submitting them will be sent in a separate email later tonight.

References

[1] Ken Goldberg, Theresa Roeder, Dhruv Gupta, and Chris Perkins. Eigentaste: A Constant Time Collaborative Filtering Algorithm. *Information Retrieval*, 4(2), 133-151. July 2001.

[2] Ken Goldberg, Anonymous Ratings Data from the Jester Online Joke Recommender System, http://www.ieor.berkeley.edu/~goldberg/jester-data/.