INFO371 Problem set 4: k-NN, TF-IDF

Your name:

Deadline: Tue, Feb 25th midnight

Introduction

This is the first homework where you have to implement yourself some of the common ML algorithms. In particular, you will program k-NN with cosine similarity, using both bag-of-words and TF-IDF approach. Before you start, ensure you have read:

- Daume "A course in machine learning", Ch 3.1–3.3. This a good introduction to k-NN.
- my lecture notes, titled *machineLearning*, the chapter about k-NN. This includes metric distance, cosine distance, and TF-IDF. Feel free to give some feedback so I can improve on what is unclear. (The currect version is a quick hack.)

You will use two types of data: first, various texts from Cantenbury corpus with several books added from Project Gutenberg, and thereafter Rotten Tomatoes, brief movie reviews. In case of the texts, your task is to find the correct source of the text, in case of tomatoes, to predict if the review is for a "rotten" or "fresh" movie.

There is also a separate file with some example python code where you can find chunks of useful code for this problemset.

Please submit a) your code (notebooks, rmd, whatever) and b) the lab in a final output form (html or pdf). Unlike PS1, you are free to choose either R or python for solving this problem set.

While some of the intermediate output may be informative, please don't include too many lines of it in your solutions!

Note: it includes questions you may want to answer on paper instead of computer. You are welcome to do it but please include the result as an image into your final file.

Working together is fun and useful but you have to submit your own work. Discussing the solutions and problems with your classmates is all right but do not copy-paste their solution! Please list all your collaborators below:

1.
2. ...

1 Where are these texts coming from?

The data file *texts.csv* contains the texts you have to classify. It contains the following variables:

name name of the original file. It is usually *author-name* form and it should be fairly easy to find the original text in most cases.

size size of the original text, in bytes

lines size of the original text, in lines

chunkid chunk id, from 1 and growing, see *chunk*. If you want to re-assemble the original texts, you just have to put these next to each other in the order of *chunkid*.

chunk a page of text. It is not really a page, just 25 lines of text, whatever happened to be on those 25 lines.

Text chunks are just verbatim texts that may contain all kind of characters, including newlines. Note the file is tab separated and uses quotes for strings.

Your task is to read all the texts, convert these to a) bag-of-words, and b) TF-IDF-s, and predict the correct source using k-NN.

1.1 Bag of words

First, let's use bag-of-words (BOW) approach.

1. Load the data. Print out a few lines of it to inspect it's structure.
2. Inspect some of the texts. Note that *chunkid* 1 corresponds to the first page of the text.
3. List all the text sources listed in variable *name*. How many different texts does the data contain?

When you feel you know the data well enough, it's time to split it into testing and validating parts.

Warning: start slow. The dataset contains 13,000 texts and 65,000 unique words. If you load in everything, the BOW matrix takes approximately 8GB of RAM. I recommend to start with very small datasets, say 10 pages for both training and validation. When your code works, start increasing the size. Stop where it gets too slow, you don't have to run everything (it takes 18G RAM). But note that if you drastically scale down the amount of data, you should also select only a few sources. Otherwise you may have no examples from certain sources to compare to.

4. Now when you have created your training and testing sets, it is time to create the dictionary. See the example code for python examples, there are many examples on the web about how to do it with R (see text mining with R or text2vec package).

   This process contains two steps: 1) create a dictionary of all your texts; 2) recode all the texts as BOW vector. See the lecture notes for explanation about BOW.

   Note you may want to feed in all your data, i.e. both training and validation data into the dictionary. This ensures all words in your validation are represented in the dictionary.

5. Now you have all your (training and validation) texts in BOW form. This is great, we just transformed texts into (numeric) vectors! Now implement cosine similarity between these vectors.

   Ensure you have read the lecture notes about cosine similarity. Write a function that takes in two vectors, $\mathbf{x}$ and $\mathbf{y}$, and returns the corresponding cosine similarity $\mathbf{c}(\mathbf{x}, \mathbf{y})$. Test if $\mathbf{c}(\mathbf{x}, \mathbf{x}) = 1$, test also a few other vectors.

6. Finally, implement k-NN. I mean implement it yourself, don't use pre-existing libraries. I recommend to do 1-NN first, and when this works to extend it to k-NN. You need an algorithm along these lines:

   (a) Pick a validation case $\mathbf{y}$ (later you will loop over all of these).

   (b) for each vector in the training set $\mathbf{x_i}$, compute the cosine similarity $\mathbf{c}(\mathbf{y}, \mathbf{x_i})$. Store this number, and ensure you know which $\mathbf{x_i}$ corresponds to each $\mathbf{c}$ value.

   (c) Now order all the cosine similarities you just computed in an increasing order.

   (d) Pick the $\mathbf{k}$ smallest $\mathbf{c}$-s. These correspond to your $\mathbf{k}$ nearest neighbors! Ensure you know which texts these are.

7. Now you know your nearest neighbors. Organize a majority voting among them so you will learn which text is the most popular among them. I recommend to create a frequency table and pick the most common text source name based on this.

8. Compute accuracy (percentage of correct predictions). How good is your algorithm?

Now it is time to start increasing the training and testing sets. I repeat: go slow. Training set of 1000 and testing set 100 should be fine. Go further if your memory and speed permit. When you have found the limits you don't want to exceed (say, 5 mins for the run), it is time to play with k.

9. compare different k values. 1, 5, 25 are a good choice. Which k gives you the best performance? What is your highest accuracy?

1.2 TF-IDF

BOW is great but could be even greater. Can we get a better result using TF-IDF? TF-IDF is simply a way how to weigh the word frequency in a more informative way.

1. implement TF-IDF transformation. Note: implement it yourself, don't rely on existing libraries! This involves manipulating your training and validation data matrices, nothing else needs to be done. Example will be given in the notes.

2. Now ensure that your cosine similarity you implemented earlier also works for vectors in TF-IDF form. It should, without any modifications, if you implemented in well.

3. Run your k-NN with cosine similarity algorithm again, using several k values, like 1,5,25. The algorithm should not need any modifications.

4. Finally, comment your results. How accurate is BOW versus TF-IDF? How does choice of k change the results? Is BOW or TF-IDF faster to run?

2 Are tomatoes fresh or rotten?

Our next task is to use your freshly minted methods for classifying the Rotten Tomatoes movie reviews. Please familiarize yourself a little bit with the webpage. Briefly, approved critics can write reviews for movies, and evaluate the movie as "fresh" or "rotten". The webpage normally shows a short "quote" from each critic, and whether it was evaluated as fresh or rotten. You will work on these quotes below.

The central variables in *rotten-tomatoes.csv* are the following:

critic name of the critic

fresh evaluation: 'fresh' or 'rotten'

quote short version of the review

review_date when the review was written.

There are more variables like links to IMDB.

1. Load the data. Inspect it a little bit to see how it is coded and organized. How many cases do you have?

2. Clean the data. Retain only cases where *fresh* and *quote* are present and non-empty. Remove repeated observations (there are such in data). How many cases will be left?

3. Select training and validation data. You would like to split all cases into 80-20 groups. However, as before, this may be slow. Start slow with perhaps 100 random quotes split into two groups.

4. Your task is to find the closest training quotes for each test quote, and based on those predict if the movie is fresh or rotten according to the quote.

   Follow the same steps you did with the texts above:

   (a) create dictionary and BOW of all quotes

   (b) run k-NN with several different k-s, and predict if fresh or rotten. In each time compute the accuracy.

   (c) transform your data into TF-IDF form and repeat k-NN.

   (d) inspect a few cases where the tomato was correctly/incorrectly predicted. Can you explain why the algorithm behaved in the way it behaved?

5. Finally, comment your results. What worked better: reviews or text pages? What worked better: BOW or TF-IDF?