Data Mining

Homework 3

Due: 29/11/2015, 23:59.

Instructions

You must hand in the homeworks electronically and before the due date and time.

Handing in: You must hand in the homeworks by the due date and time by an email to the instructor that will contain as attachment (not links!) a .zip or .tar.gz file with all your answers and subject
`[Data Mining class] Homework 3`
After you submit, you will receive an acknowledgement email that your homework has been received and at what date and time. If you have not received an acknowledgement email within 2 days after you submit then contact the instructor.

The solutions for the theoretical exercises must contain your answers either typed up or hand written clearly and scanned.

The solutions for the programming assignments must contain the source code, instructions to run it, and the output generated (to the screen or to files).

For information about collaboration, and about being late check the web page.

Most of the questions are not very hard but require time and thought. **You are advised to start as early as possible, to work in groups, and to ask the instructor in case of questions.**

Problem 1. Consider the problem of classifying the MNIST digit recognition data, which we have used for k-NN classification in the notebook `Intro-to-Classification.ipynb`. Build a nearest-neighbor classifier using the full dataset and then using SVD-based dimensionality reduction. Plot the accuracy of your classifier as a function of the dimensionality of the data using k-fold cross validation. For your experiments, use projections of the dataset into $d' = 1,2\ldots,d$ dimensions, where d is the original dimension of the dataset. Also show the accuracy of the classification in the original (nonprojected) data. Repeat the experiment for $k = 1,2,5$ and 10 neighbors.

Show your work in an ipython notebook.

Problem 2. For this exercise we will implement some versions of the A-priori algorithm.

1. Implement the simple A-priori algorithm in Python. For every round you have to read the file and you can keep in memory the frequent elements that you find.
   - You can store the itemsets during the computation in standard Python data structures (lists, sets, dictionaries) without doing the fancy stuff that we have seen.
   - The input format is one line per basket, and each line contains the IDs of the items in the basket separated by space.
   - You can assume that the IDs are $0, 1, 2, \ldots$.
   - At the end save the itemsets that you compute in a file, one line per itemset, each line containing the items in the itemset separated by space.
   - Report the total running time, and the number of itemsets found.
   - You will probably find very useful the `itertools` package.

2. Implement the simple, randomized algorithm of Section 6.4.1 of the book.
   - Read the file once, sample baskets into memory, and then compute the itemsets just by reading the memory.
   - Implement the two techniques of Section 6.4.2 to remove false positives and reduce false negatives.
   - At the end save the itemsets that you compute in a file, one line per itemset, each line containing the items in the itemset separated by space.
   - Report the total running time, and the number of itemsets found, both before and after eliminating the false positives.

After making sure that your implementations are correct (by running them on some toy examples that you create) run them on two datasets:

1. First, on the file `http://aris.me/contents/teaching/data-mining-2015-fall/homeworks/retail.dat.` which contains the (anonymized) retail market basket data from an anonymous Belgian retail store. Use as threshold $t = 500$, and as sampling probability $p = 0.1$.
2. Then try them on the bigger dataset `http://fimi.ua.ac.be/data/webdocs.dat.gz`, which was built from a spidered collection of web html documents. Use as threshold $t = 500{,}000$ and as sampling probability $p = 0.0001$. This example shows why we often prefer approximate results: here the standard algorithm may be very slow (hours), whereas a good implementation (but without any smart techniques) of the randomized algorithm should be able to produce results in a few minutes.

For each of the datasets compare the results of the two algorithms. First write a small program to check that the randomized algorithm does not return itemsets that are not produced by the simple algorithm. Also compare the full results with the results that you obtain with the randomized algorithm if you set that threshold in the sampled set to (i) tp, and (ii) $0.9tp$. More specifically, report how many frequent itemsets does the randomized algorithm return for each of the two thresholds and how much time it needs.

Problem 3. When building a decision tree, we select the best split node using an impurity measure. An example of impurity measure is the *entropy*. Consider node t in the decision tree and let $p(i \mid t)$ be the fraction of the records associated with node t and belonging to class i. Then, if there are c classes in total, we measure the impurity of t using entropy as follows:

$$H(t) = -\sum_{i=1}^{c} p(i \mid t) \log p(i \mid t).$$

1. Consider a node t in the decision tree that corresponds to a continuous feature (e.g., the salary). Assume that you want to partition the points that are in node t using k salary ranges $R_1, \ldots, R_k$ that are contiguous, non-overlapping and cover the same total salary range as t. Design an algorithm that finds these ranges and creates nodes $t_1, \ldots, t_k$ such that node t_i corresponds to range R_i and

   $$H(t_1) + H(t_2) + \ldots + H(t_k)$$

   is minimized.
2. Compute the running time of this algorithm as a function of the number of points n_t that are associated with node t.