CSE 158, Fall 2017: Homework 3

Instructions

Please submit your solution **by the beginning of the week 7 lecture (Nov 13).** Submissions should be made on **gradescope**. Please complete homework **individually**.

These homework exercises are intended to help you get started on potential solutions to Assignment 1. We'll work directly with the Assignment 1 dataset to complete them, which is available here:
`http://jmcauley.ucsd.edu/data/assignment1.tar.gz`

Executing the code requires a working install of Python 2.7 or Python 3.

You'll probably want to implement your solution by **modifying the baseline code provided**.

Note that you should be able to join the competitions using a UCSD e-mail. The competition pages can be found here:
`https://inclass.kaggle.com/c/cse158-258-fa17-visit-prediction`
`https://inclass.kaggle.com/c/cse158-fa17-categorization`
`https://inclass.kaggle.com/c/cse258-fa17-rating-prediction`
though you will need to use the login keys to access the competition:
visit prediction: `https://www.kaggle.com/t/f5cb34fc0265469eb7c4d4e86618e650`
rating prediction: `https://www.kaggle.com/t/a9061008f33740a6a0b8e120b775bc7b`
category prediction: `https://www.kaggle.com/t/9107d033f5bb4135a8aa5bd4cf0ecfba`

Please include the code of (the important parts of) your solutions.

Tasks (Visit prediction)

First, since the data is quite large, when prototyping solutions it may be too time-consuming to work with all of the training examples. Also, since we don't have access to the test labels, we'll need to simulate validation/test sets of our own.

So, let's split the training data ('train.json.gz') as follows:
(1) Reviews 1-100,000 for training
(2) Reviews 100,001-200,000 for validation
(3) Upload to Kaggle for testing only when you have a good model on the validation set. This will save you time (since Kaggle can take several minutes to return results), and also will stop us from crashing their website...

1. Although we have built a validation set, it only consists of positive samples. For this task we also need examples of user/business pairs that *weren't* visited. Build such a set by randomly sampling users and businesses until you have 100,000 non-visited user/business pairs. This random sample combined with your 100,000 validation reviews now corresponds to the complete validation set for the visit prediction task. Evaluate the performance (accuracy) of the baseline model on the validation set you have built (1 mark).

2. The existing 'visit prediction' baseline just returns *True* if the business in question is 'popular,' using a threshold of the 50th percentile of popularity (`totalVisits/2`). Assuming that the 'non-visited' test examples are a random sample of user-visit pairs, is this particular threshold value the best? If not, see if you can find a better one (and report its performance), or if so, explain why it is the best (1 mark).

3. Users may tend to repeatedly visit business of the same type. Build a baseline that returns 'True' if a user has visited a business of the same category before (at least one category in common), or zero otherwise (1 mark).[1]

4. To run our model on the *test* set, we'll have to use the files 'pairs_Visit.txt' to find the userID/businessID pairs about which we have to make predictions. Using that data, run the above model and upload your solution to Kaggle. Tell us your Kaggle user name (1 mark). If you've already uploaded a better solution to Kaggle, that's fine too!

Tasks (Category prediction)

First, build training/validation sets consisting of only those reviews that have a 'categoryID' field. These are reviews that had exactly one of the 10 categories selected, so that there is no ambiguity in their category

[1]Here consider the 'category of a business' to be the set of all categories users selected for that business.

label (the test reviews also have only a single, unambiguous category). For this question you may also wish to sub-sample your data (e.g. by taking a random 10%) if experiments are taking too long to run.

5. A trivial predictor might assume that each user tends to visit only one category of business. For each user in the training set, identify the category of business they visit most frequently (in the case of a tie, choose whichever category is more popular overall). Then, for each review in your validation set, simply return whichever category you identified as being the most popular (otherwise return 0 if you've never seen the user before). Report the performance of this classifier on the validation set (1 mark).

Next, we'll try looking for the presence of common words. Start by removing punctuation and capitalization, and finding the 500 most common words across all reviews ('reviewText' field) in the training set. See the 'text mining' lectures for code for this process.

6. The most common words probably won't do much to help us predict categories. Instead, let's find which words are *more* common in each category compared to the others. First, compute the frequency of those 500 words you just found as follows:

$$frequency_{\text{word}} = \frac{\text{number of times the word appears}}{\sum_{i=1}^{500} \text{number of times the } i^{\text{th}} \text{ most popular word appears}}$$

Next, compute the above frequency *per category*, e.g.:

$$frequency[Bar]_{\text{word}} = \frac{\text{number of times the word appears in 'Bar' reviews}}{\sum_{i=1}^{500} \text{times the } i^{\text{th}} \text{ most popular word appears in 'Bar' reviews}}$$

Having computed this we can find which words are *more* popular in one category compared to their overall frequency across all categories, i.e.,

$$frequency[Bar]_{\text{word}} - frequency_{\text{word}}$$

Report for each of the 10 categories the 10 words that are *more* frequent in that category compared to other categories (1 mark).

Next, let's run some classifiers by building a simple feature vector out of our 500 most popular words as follows:

```
[commonWords[0] in review, commonWords[1] in review, ..., commonWords[499] in review]
```

7. Train an SVM to distinguish *Bars and non-Bars only*. That is, discard all other categories from the training set in order to train a binary classifier, but keep them in the validation set (your classifier will simply be wrong for those instances). Train for $C \in \{0.01, 0.1, 1, 10, 100\}$ (the regularization parameter for the SVM) and report the best performance you obtain on the validation set (1 mark).

8. **(Hard)** Rather than training a binary SVM as we did above, train *10* SVMs, that distinguish each category from all other categories (i.e., one which distinguishes Bars from all other categories, one of which distinguishes American Restaurantsn from all other categories, etc.). Again select $C \in \{0.01, 0.1, 1, 10, 100\}$ for each classifier using the validation set. For the validation set, classify each point according to whichever of the ten classifiers is *most confident* about the label being positive (see 'decision_function' in the SVM) library. Report the performance of this classifier on the validation set, and upload the solution to Kaggle by running it on the test data (1 mark).