BANA 290: Machine Learning for Text: Spring 2018

Homework 3: Unsupervised Learning

Conal Sathi and Sameer Singh (with help from Yoshitomo Matsubara)
https://canvas.eee.uci.edu/courses/9097

The third programming assignment will familiarize you with the use of unsupervised methods for learning text representations. In particular, we will be comparing sparse vectors with word embeddings, and their use for K-Means clustering. The submissions are due by midnight on **May 29, 2018**.

Task: Sentiment Classification

Although the goal here is to also perform sentiment classification, like the first two assignments, we will be exploring the dataset using unsupervised learning techniques like clustering and word embeddings. You will be using the same dataset containing the text of the movie reviews from IMDB.

Data: The data is the same as the second homework, i.e. available on the Kaggle website[1]. The file is called `data.zip`, which contains similar structure as the data for the first homework. Unzip `data.zip` to a local folder called `data/`.

Pretrained Word Embeddings: You are also provided a file containing pretrained word embeddings, called `glove.6B.100d.word2vec_format.txt.zip`[2]. Unzipping this file will give you a single large file (330MB) called `glove.6B.100d.word2vec_format.txt`, which you should put in the same folder as the code. If you open it (using something that can handle large files), you will see it simply contains 100 numbers for each word, i.e. its embedding vector, for 400,000 words, sorted by their frequency.

Source Code Code is available for you to develop your submission, at https://canvas.eee.uci.edu/courses/9097/assignments/182968. There are two main files, `compute_features/py` and `clustering.py`, that you will need to run. First, execute:

```
python3 compute_features.py
```

You can also use PyCharm, etc. This does a number of things for you (you should be able to understand the code):

- Use pretrained word embeddings to compute a vector for each review in the training data, by averaging the embeddings of the words that appear in the review. To be able to use in other scripts, the script saves these features to a file called `reviews_features_dense_pretrained.pkl`.
- Learn custom word embeddings based on the reviews in the training data, using parameters defined in `get_custom_word_embeddings()`. Feel free to change these parameters in the course of the homework. These learned word embeddings are saved in a file called `custom_word_embeddings.pkl`.
- Use these *custom* embeddings to compute a feature vector for each review, which is just the average embedding of the words in the review. This is different from pretrained embeddings since here we are using the word embeddings that you learned on this dataset. The feature vectors of the reviews are stored in a file called `reviews_features_dense_custom.pkl`.
- Finally, the code also writes out a simple `TfidfVectorizer`-based sparse vector for each review to a file called `reviews_features_sparse.pkl`. Again, feel free to change the default vectorizer, if you feel it will have a significant impact on the results.

The second file, `clustering.py`, trains a K-nearest neighbor classifier based on the parameters. The main parameter is `-features`, which allows you to specify which type of feature (`sparse`, `pretrained`, or `custom`) will be used for clustering. The second parameter is `-k`, is optional, and runs the code in two modes.

1. When the `-k` is omitted and the code it executed:

   ```
   python3 clustering.py -features <put feature type here>
   ```

   it trains a bunch of classifiers for different Ks, and outputs the *intertia vs K* graph that can be used to select the K (as we talked about in class, using the *elbow* method). Feel free to modify the range of K (in `<feature type>_plot_k_vs_inertia()`) and other parameters of the classifier, like the metric, in `clustering()`.

[1] https://www.kaggle.com/c/sentiment-analysis-sp18/
[2] Description of research here: https://nlp.stanford.edu/projects/glove/

2. Once you have identified the K, and other parameters, for the features, you can include `-k` and execute:

```
python3 clustering.py -k <put K here> -features <put feature type here>
```

This time, it merely trains a single classifier with the K you have chosen, and prints out its purity on the training data. It also writes out the clusters into a file called `<feature type>_clusters.csv`, describing the cluster assignment of every review in the training data.

Word Embeddings

The first part of the assignment requires and investigation of the similarities and differences between different word and document representations.

Words First, you will need to compare and contrast pretrained and custom word embeddings in neighborhood they capture. Once you are happy with the parameters of the custom word embeddings in `compute_features.py`, you can load the embeddings and run nearest neighbor queries, as shown in the included Jupyter notebook `word_embeddings.ipynb`. Use this to explore words that highlight the similarity and differences between the two word embeddings. You can also encouraged to use other handy functions in `gensim` like `closer_than`, `distance(s)`, `rank`, `n_similarity`, `doesnt_match`, etc.[3] If you want to plot some words in 2D to show the difference, refer to the in-class notebook available here: https://canvas.eee.uci.edu/courses/9097/assignments/182980.

Reviews There are now three options for vectors for the reviews: sparse vectors, dense vectors using pretrained embeddings, and dense vectors using custom embeddings. In order to compare these three representations, we will primarily rely on nearest neighbors, i.e. you will show how these different representations result in different neighbors. The code for making these queries is available in the notebook as well, but use your intuitions and insights from the differences in word neighbors above to guide your choices. You are welcome to change the parameters of the nearest neighbor search, such as the metric, etc.

Clustering

You will now explore these unsupervised methods in their application to K-Means clustering on the movie reviews. Note that the primary objective here is for you to identify and explore what the clustering is doing, and not necessarily maximize accuracy.

Picking a K For each of the three types of review features (sparse vectors, dense pretrained vectors, and dense custom vectors), call the clustering code without a K value to get the K versus inertia plots. Change the parameters of this search, and the classifier, till you are happy with the results. Include the three plots, and describe which K (and why) you picked for each of the three features.

Comparing Clusters For each of the three types of review features (sparse vectors, dense pretrained vectors, and dense custom vectors), we have a `clusters.csv` file that contains the predicted clusters. Use this file, and the corresponding files in the data folder, to explore some of these clusters. Pick 1-2 clusters from each clustering approach to describe what you think each cluster is capturing. Also describe the differences you see between the different representations, building upon the insights you have from comparing words and reviews in the question above. You might have to write some code to automatically print reviews in the same cluster.

Report the Purity Show a table with the purity that you obtained by the nearest neighbor classifier with each of the methods, and describe whether they lined up with what you found about the clustering earlier.

[3] See documentation here: https://radimrehurek.com/gensim/models/keyedvectors.html#gensim.models.keyedvectors.WordEmbeddingsKeyedVectors.

What to submit?

Prepare and submit a single write-up (**PDF, maximum 3 pages**) and all modified Python source code (`clustering.py`, `word_embeddings.ipynb`, and `compute_features.py`), compressed in a single `zip` file, to Canvas. **Do not include your student ID number**, since we might share it with the class if it's worth highlighting. The write-up and code should address the following.

1. **Word Embeddings, 35 points** For both of these, the primary submission will be changes in the notebook. In the writeup, refer to the output in the notebook to describe your insights, in a few sentences for each.
   (a) *Comparing Words, 15 points:* Use the `gensim` utility functions to show the similarities and differences between the two sets of embeddings, using 3-4 words and their neighbors.
   (b) *Comparing Reviews, 20 points:* Use the notebook to identify a few neighboring reviews that again, highlight the similarities and differences between all *three* types of review representations.

2. **Clustering, 60 points** For clustering, you will be including plots and examples in the report directly, and thus a majority of the report will focus on the clustering.
   (a) *Picking K, 15 points:* For each of the three feature types, include the K versus Inertia plots, and describe how you selected the K for them.
   (b) *Showing Clusters, 35 points:* Give at least 2 clusters for each of the three methods, with 2-3 reviews in each cluster, and describe in a sentence or two what you think the clusters represent. Select clusters such that you can also write in a few sentences, at the end, how these clusters show the differences between the different representation methods.
   (c) *Purity, 10 points:* Include the purity for the three feature types, and describe in a sentence or two whether they match your expectations, and why or why not.

3. **Statement of Collaboration, 5 points**

Statement of Collaboration (5 points)

It is **mandatory** to include a *Statement of Collaboration* in each submission, with respect to the guidelines below. Include the names of everyone involved in the discussions (especially in-person ones), and what was discussed.

All students are required to follow the academic honesty guidelines posted on the course website. For programming assignments, in particular, we encourage the students to organize (perhaps using Piazza) to discuss the task descriptions, requirements, bugs in our code, and the relevant technical content *before* they start working on it. However, you should not discuss the specific solutions, and, as a guiding principle, you are not allowed to take anything written or drawn away from these discussions (i.e. no photographs of the blackboard, written notes, referring to Piazza, etc.). Especially *after* you have started working on the assignment, try to restrict the discussion to Piazza as much as possible, so that there is no doubt as to the extent of your collaboration.