HW1: Amazon Review Classification

Bhavika Tekwani
btekwani@gmu.edu

Introduction

The crux of this Amazon Review Classification problem is implementing the Nearest Neighbour algorithm but it presents the opportunity to choose from several information retrieval methods for text mining. The solution I implemented was a TF-IDF based model that vectorizes each of the 18506 Amazon reviews in train and test datasets and then uses Nearest Neighbour to classify them as positive (+1) or negative (-1).

Rank & Accuracy

As of 09/19/2016 11:16 AM, the public accuracy score is 0.56 and rank is 23.

Team name

bolt79

Approach

Natural Language Processing: I used NLTK for preprocessing the train and test datasets. At the preprocessing stage, I chose to use lemmatization over stemming even though both these forms of term normalization yield similar results and depend more on the identification of “root” words rather than linguistic usage in a particular text document.

Term Frequency-Inverse Document Frequency (TF-IDF): Sentiment analysis in this problem is reduced to knowing which words occur most frequently in positive reviews and which ones occur most frequently in negative reviews. TF-IDF involves two steps – one is computation of a normalized Term Frequency that is indicative of how often a word appears in a document. Since text is often unstructured, the term frequency of a word may change depending on the length of the document it appears in. For example, a word may appear more frequently in a longer document than a shorter one.

TF(t) = (number of times term t appears in a document) / (total number of terms in the document)

Inverse document frequency measures how important a term is. To avoid one pitfall of IDF calculations which is measuring the influence of stop words like “is”, “this”, “a”, “the”, “and”, etc., I have filtered out stop words from the corpus in the natural language processing stage. I intended to reduce the number of dimensions and calculations involved by reducing the vocabulary in this way. IDF is calculated as follows:

IDF (t) = $\log_{10}$(total number of documents) / (number of documents with term t in them)

TF-IDF vectorization of train data would give me 18506 vectors containing TF-IDF weights for each unique word in the training corpus.

	0	**1**	**2**	**3**	**4**	**5**	**.....**	**n**

0	0.456	0.0012	0.005	0.3	0.0001	0.1	.....	0.232
:	:	:	:	:	:	:	:	:
18506	0.22	0.007	0.065	0.8	0.336	0.4785	.....	0.009

Fig.1 Representation of a TF-IDF model for train data

The immediately apparent drawback of this method is that the dimensions of the vectors increase with the size of the vocabulary. With 18506 reviews in the train file and an equal number in the test file, the size of the vocabulary including unique terms from both in our case was 36956.

To make similarity calculations between matrices of such sparsity feasible, I used Latent Semantic Analysis to reduce each review to two components.

Latent Semantic Analysis (LSA): LSA converts term document matrices to semantic spaces of lower dimensionality. LSA is a technique used to combat synonymy and polysemy in document classification, clustering and information retrieval. Synonymy refers to the phenomenon where different words are used to describe the same idea. Polysemy refers to the phenomenon of a word having multiple meanings across subject and usage contexts. The assumption in LSA or similar dimensionality techniques is that the representation obtained is the “true” representation and it uncovers the underlying similarity between two documents which have no common terms. A disadvantage of LSA is that a significant amount of information is lost when term documents are reduced to a small number of components. Singular Value Decomposition (SVD) is used to extract features from documents in my solution.

Nearest Neighbour Algorithm: The variant of the Nearest Neighbour algorithm I've implemented is called k-Nearest Neighbour (kNN). The kNN assigns a class label to an unseen data instance based on the class labels of the nearest 'k' seen instances. This is called majority voting. The similarity metric used is cosine similarity. Cosine similarity is calculated using L2 normalization as a parameter for SVD while creating the LSA pipeline in my solution. The computational complexity is important while setting a while for 'k'. A general rule of thumb seems to be that we can start with $k = n^{1/2}$ for n samples, which would be k = 136 for our 18506 samples. I've dealt with the problem of breaking ties during majority voting by selecting odd values of k while testing for the optimal k. In my last submission, k=379.

References

Manning, Christopher D., Prabhakar Raghavan, and Hinrich Schütze. "Scoring, Term Weighting and the Vector Space Model."*Introduction to Information Retrieval*. New York: Cambridge UP, 2008. N. pag. Web.

Manning, Christopher D., Prabhakar Raghavan, and Hinrich Schütze. "Matrix Decompositions and Latent Semantic Indexing." *Introduction to Information Retrieval*. New York: Cambridge UP, 2008. N. pag. Web.

Alex9311. "Methods of Calculating Cosine Similarity between TF-IDF Vectors." *StackOverflow*. N.p., 2 Feb. 2016. Web. 19 Sept. 2016. **<http://stackoverflow.com/questions/35152850/methods-of-calculating-cosine-similarity-between-tf-idf-vectors>**.

Null-Hypothesis. "Tf-idf-cosine: To Find Document Similarity." *StackOverflow*. N.p., 25 Aug. 2012. Web. 19 Sept. 2016. <http://stackoverflow.com/questions/12118720/python-tf-idf-cosine-to-find-document-similarity?rq=1>.

Duran, Fletcher. "Dealing with Ties, Weights and Voting in KNN." *StackOverflow*. N.p., Dec. 2010fle. Web. 19 Sept. 2016. <http://stats.stackexchange.com/questions/45580/dealing-with-ties-weights-and-voting-in-knn?rq=1>.