CS 535: Mini-Project 1

Sakib Jalal

September 28, 2017

Abstract

Instructions for running the code for this assignment can be found in README.md.

1 Problem 1: Apriori

1.1 Error in Code

When running the Apriori algorithm, the vector representation showed 9 baskets with their respective item vectors, but the dataset provided had 10 baskets, so one of them was not being displayed. To fix this, I found that there was a loop on line 40 that goes through the length of the dataset of baskets and builds a matrix (the matrix to be displayed), but this loop was going from 0 to $len(data) - 1$, so it was only looking at the first 9 baskets. I removed the -1 from the code and the matrix representation then showed 10 baskets as expected.

1.2 Market Basket Analysis

I used the Apriori algorithm to analyze a data set of $N = 8993$ questionnaires filled out by shopping mall customers in the Bay Area. Each observation in the demographics data had 14 values. I filtered out observations with missing values was left with $N = 6876$ observations. Each predictor had between 2 and 9 values, so I wanted to create one dummy variable per possible value, for a total of 83 dummy variables.

To do this, I used an additive array (CCL) of length 14 in order to define a unique range of values for each variable. CCL was $[9, 2, 5, 7, 6, 9, 5, 3, 9, 9, 3, 5, 8, 3]$, meaning that the first category had 9 possible values, the second had 2, and so on. I converted this into an additive sequence $[9, 11, 16, 23, 29, 38, 43, 46, 55, 64, 67, 72, 80, 83]$ to represent distinct value ranges, and then converted this into $[0, 9, 11, 16, 23, 29, 38, 43, 46, 55, 64, 67, 72, 80]$ which I directly used to shift each market basket vector into unique value ranges. This resulted in a 6876 x 83 matrix of 6876 observations on 83 dummy variables.

The Apriori algorithm found different amounts of association rules depending on the parameters provided, including support, minimum confidence, and lift. With a support of at least 10%, it found 1416 association rules involving <= 5 predictors. Here are a few things I've noted:

1. (1, (17, 35, 79), 99.58) -> If you're 18-24, in the Military, and White, the algo is 99.58% confident that the household income is between \$10,000 and \$14,999.
2. (16, (1, 68, 43, 81), 90.53 -> If your household income is between \$10,000 and \$14,999, your home is a House, you're not Married, and you most often speak English at home, the algo is 90.53% confident that you're between the ages of 14 and 17.
3. I noted that if I increased the minimum support of the consequent to over 25% or 50%, then one consequent appeared to dominate the association rules - #81, which specifies that the language spoken at home is English. This makes sense given that the questionnaires were distributed in the Bay Area.
4. I noted that if I filtered for consequents between 0 and 8 (all possible values for household income), the only consequent that showed in any rules was 1, meaning that household income is between \$10,000 and \$14,999. Assuming there isn't anything strange about the data or the algorithm, this may mean that there are often strong indicators for low income households, but less often for higher income households.

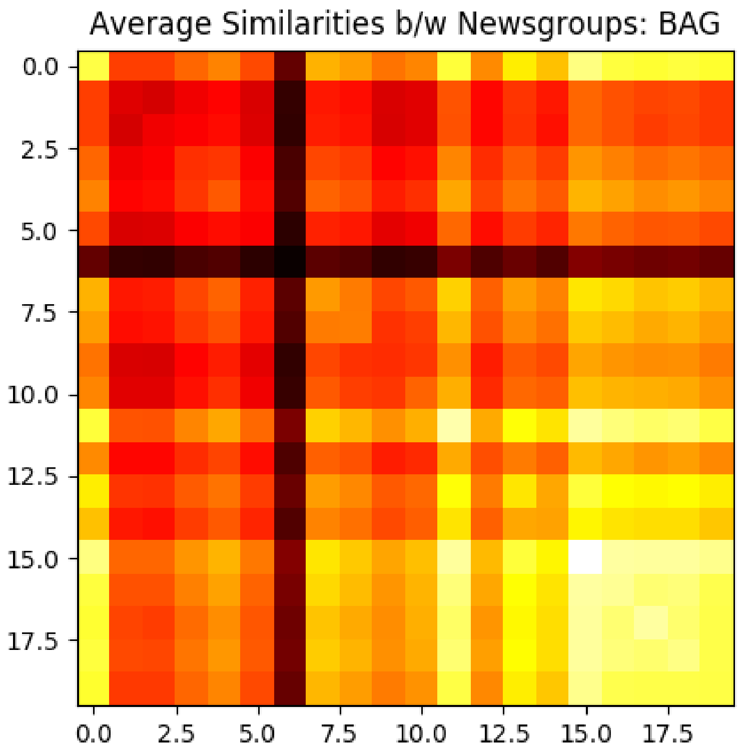
2 Problem 2: 20 Newsgroups

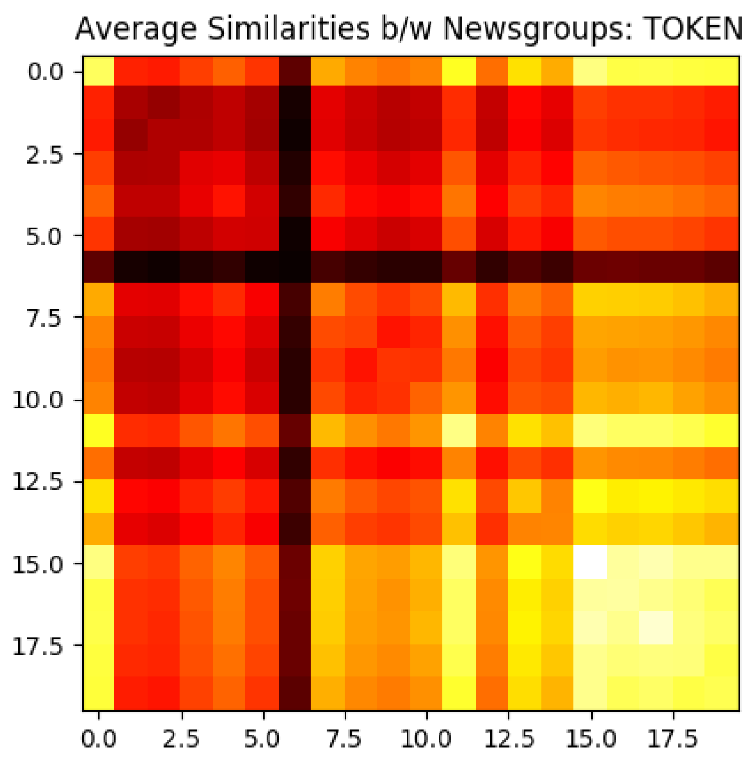
2.1 Vectorization Techniques

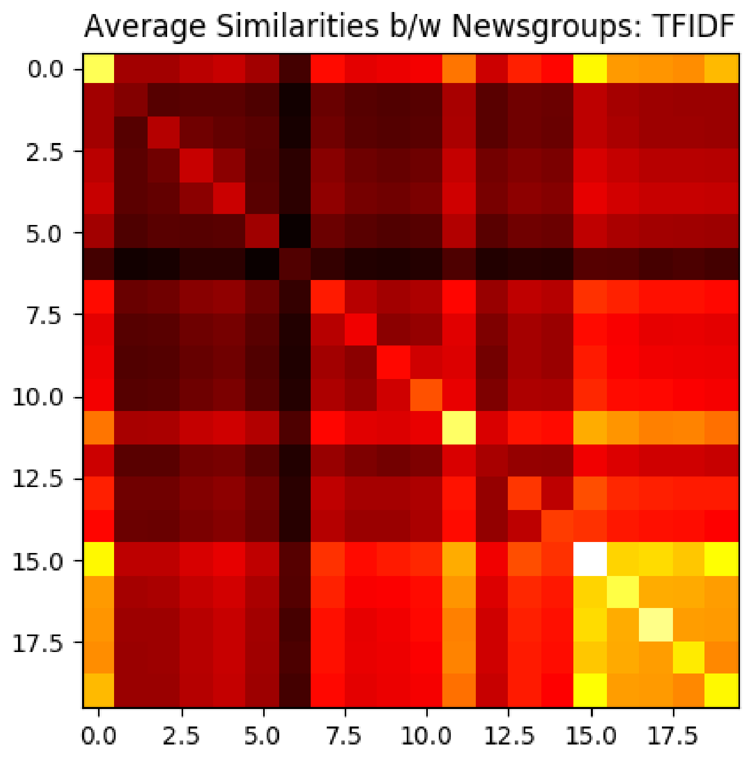
Feature vector size & number of data points (i.e., data matrix size). Sparseness of the data matrix defined as the relative average number of non-zero entries in each data vector. Potential benefit of each technique.

- Bag of Words: The feature vector is large because it needs one value for each distinct word in the entire body of documents - this is usually over 100,000. The data matrix is multiplicatively larger because it needs to store one feature vector per document in the corpus. If there are 10,000 documents, then the matrix is 100,000 x 10,000. However, the sparseness of the data matrix is huge because there are many more 0 entries (absence of word in document) than 1 entries (word exists in document). This is potentially beneficial because scikit-learn supports sparse matrix data structures which efficiently store large, sparse matrices like this.
- Tokenized Representation: The feature vector is large, but smaller than that in bag-of-words because of text preprocessing (elimination of stop words and stemming algorithms). The data matrix on the 20-Newsgroups training set was 2257 x 35788, so the number of terms went down to 35788. This matrix is also very sparse - the ratio was $365886/(2257*35788) = 0.004$, so about 0.4% of entries were non-zero. This is potentially useful because here, the data matrix is smaller than that in bag-of-words due to preprocessing and comes with a dictionary of feature indices, which correspond terms to their total number of occurrences in the corpus.
- Weighted Representation using TF-IDF Scaling: The feature vector is the same size as it was in the tokenized representation since it performs the same text preprocessing. The data matrix was also 2257 x 35788 with the same sparseness (0.4% of entries were non-zero). This technique is the most beneficial because it weights terms against the length of the document they appear in, so longer documents don't have a search advantage over shorter documents, and also weights the final score by how rare the term is in the corpus, so that rarer terms are given more weight.

2.2 Average Pairwise Similarities Among Newsgroups







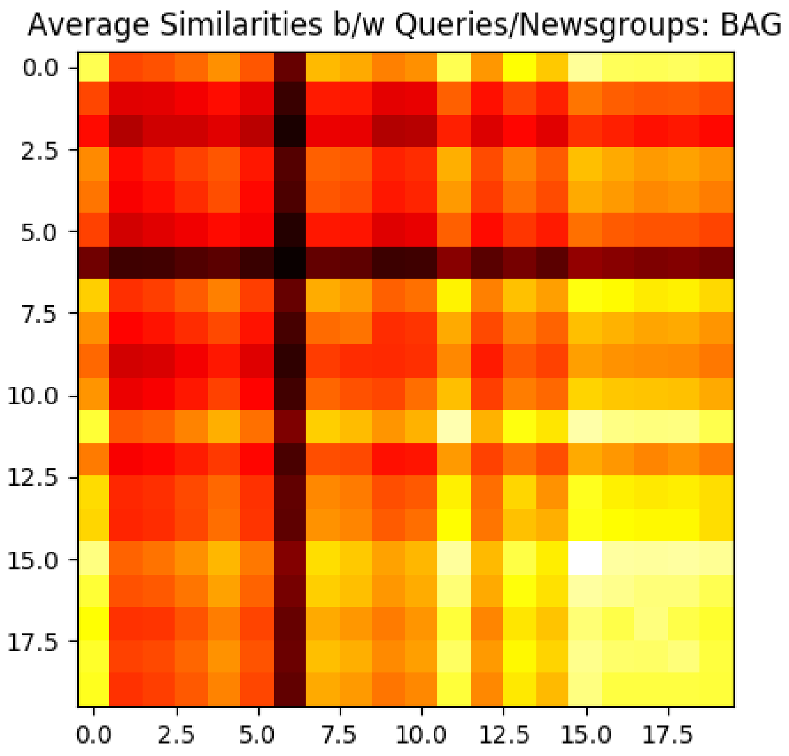
Each heat-map value within these matrices represents the average cosine similarity score between all pairs of documents within a pair of newsgroups. Darker values are lower average similarity scores and lighter values are higher average similarity scores.

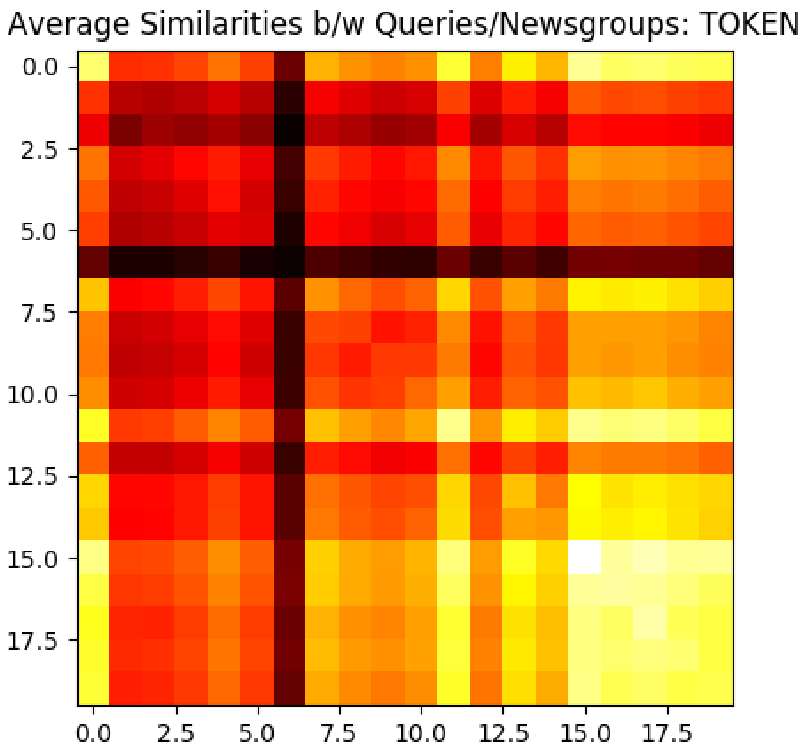
Based on these similarity scores, it seems that the highest similarity scores belonged to the pairs where a newsgroup matched with itself (see diagonal). Besides this, the highest similarity scores were between newsgroups within the range 15 to 20, and 0: "soc.religion.christian", "talk.politics.guns", "talk.politics.mideast", "talk.politics.misc", "talk.religion.misc", and "atheism". This is very interesting that these are the newsgroups with the most similar documents.

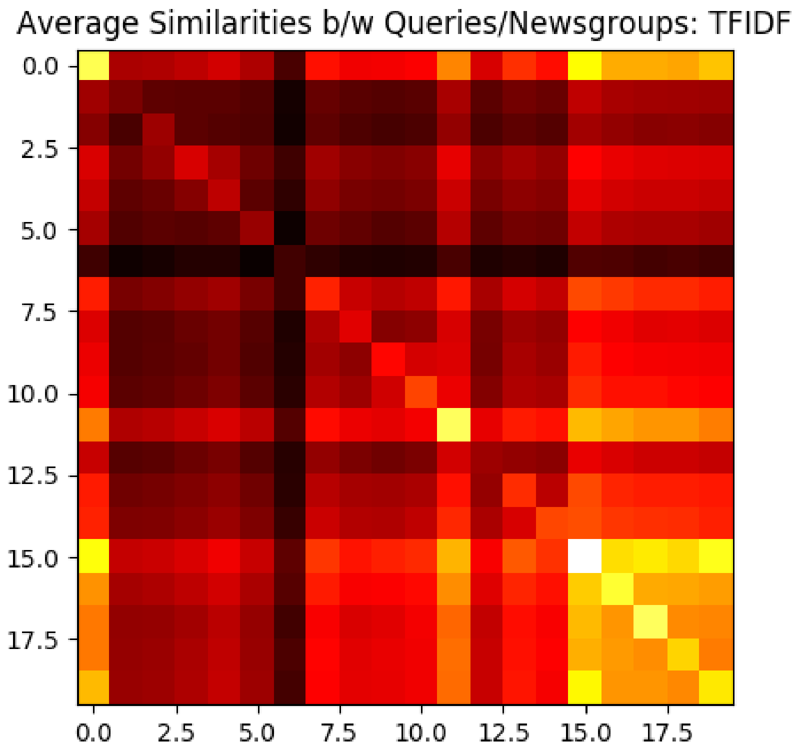
Also, it seems that the lowest similarity scores belonged to newsgroup 6, which had low scores with every single other newsgroup, and even with itself. But this also makes sense because newsgroup 6 is "misc.forsale" which by nature would contain varying contents.

The patterns of similarity scores mentioned above are consistent across data vector representations. The score plots for Bag-Of-Words and Tokenized are very similar, but the plot for Weighted TFIDF is the most appropriate. It is clearer here that the diagonal has higher similarity scores and this plot gives more information than the others since the heat variability is higher (this is because tfidf gives more weight to rarer words).

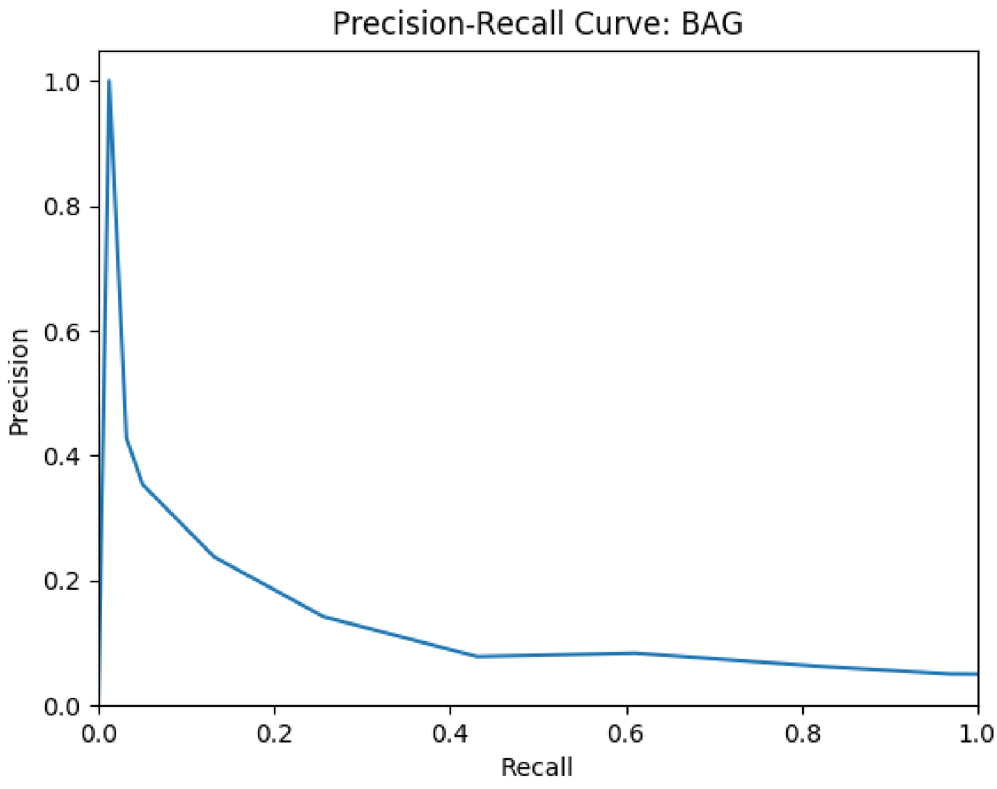
2.3 Similarity, Precision, and Recall

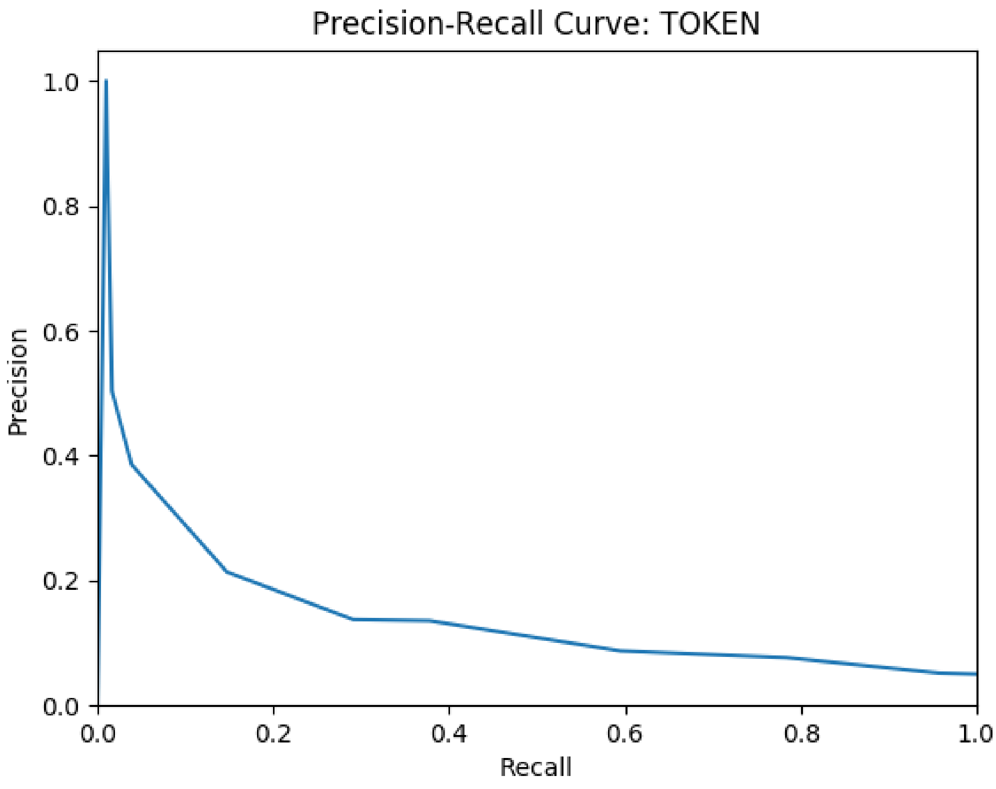


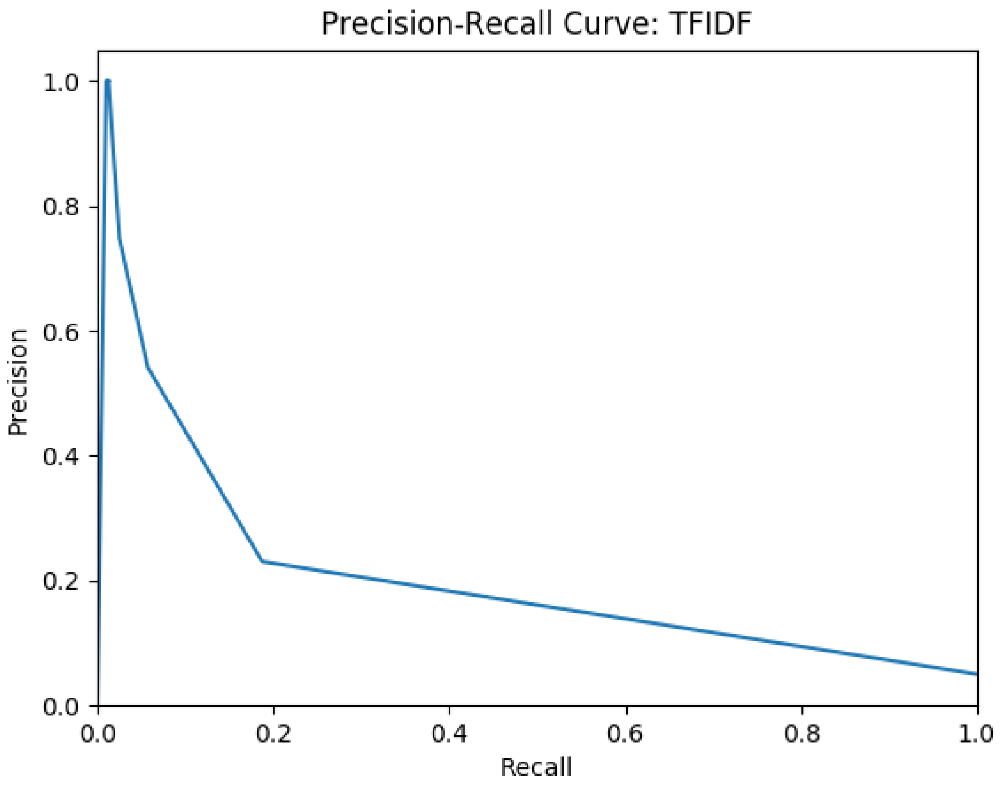




We note that these plots of average similarity scores between sets of query vectors and newsgroup representatives are not exactly but almost symmetric along their diagonal, which itself contains brighter spots (higher similarity scores) as expected, especially in the Weighted TFIDF plot. The data seems to be less noisy and have less entropy - the variability in heat by region is visibly lower. This may be because the newsgroup sets were of 100 data vectors only (rather than entire set times entire set), allowing for less variability in the cosine similarity scores along horizontal or vertical lines. This is especially pronounced in the Weighted TFIDF plot.







The three Precision-Recall Curves suggest that the most accurate vector representation of documents out of these three is TFIDF, since the area under the curve and the peak precision is the largest. Also, the previous graphs suggest that the representation of a newsgroup by the first 100 document vectors is as accurate as the representation of the same newsgroup by the full set of document vectors belonging to it. This might imply that search engines don't have to compute cosine similarities for all documents in their databases against every search query, but rather a subset of them will yield accurate enough results.

2.4 Trade-offs of Different Representations

Judging by the sizes of the sparse matrices, it seems that TFIDF and Tokenized are more space-efficient than the Bag of Words vector space representation of text. Also, the differences between the similarities of newsgroups is more pronounced in the TFIDF heatmaps, so this representation enables us to extract more meaningful information from the analysis. Finally, the precision-recall curve has the highest precision rates with the TFIDF representation, so this seems to be the most useful.