Lab 5: Classification

Introduction

The purpose of this lab was to perform Association Rule mining on the Reuters news articles dataset seen in previous labs. After the rules were created, they were to be used to classify the dataset. The results for efficiency efficacy were compared to those seen in Lab 3 with the K Nearest Neighbors, Naïve Bayesian, and Decision Tree classifiers. Additionally, a discussion on the effects of input parameters for Association Rule mining will be discussed.

Preprocessing

There were only minor surface level changes in preprocessing between this lab and the previous one on classifying. The point of this lab was to compare those classifiers with AR classification, so the feature vectors used were kept very similar. One key difference was that there was now the opportunity to handle multi-class labels. For each document with multiple classes, a new transaction was created for every topic within that transaction. For example, if a document had classes [‘earn’, ‘acq’] and keywords [‘mln’, ‘million’, ‘dlrs’, ‘may’], then two transactions would be created: [‘mln’, ‘million’, ‘dlrs’, ‘may’] => ‘earn’, and [‘mln’, ‘million’, ‘dlrs’, ‘may’] => ‘aqc’. The formatting of the actual vectors was adapted to fit into Christian Borgelt’s apriori program.

Classifying

Offline Training

The classification process began with training. To train the dataset, the vectors needed to be split into testing/training for both the topics vectors and the keyword vectors. With the input training array, the proper transactions are written to a file, and a sub process is called to execute

Christian Borgelt's apriori algorithm from the command line. This program outputs the association rules in the specified format given an appearances.txt file which constrains mining to only those rules which have topics as the consequent. Once this file is created, the rules it outputs are read in and organized into tuples of the form [class, [word_1, word_2, … word_n], support, confidence]. These tuples are then sorted primarily by confidence and secondarily by support. This completes the training of the dataset. The statistic seen later called "offline training time" refers to the total time taken to complete all of the above steps divided by the number of documents used to train the classifier.

Prediction/Online Training

Once the training is complete, the remainder of the data is fed into a function which produces predicted classes. Since the rules are already ordered by confidence and support, the algorithm works by beginning at the most confident rule and comparing the set of keywords in that rule to the set of keywords in the testing tuple to have its class predicted. If the testing keyword set is a superclass of a rule keyword set, then the class label associated with that rule is decided as the predicted label for the document. If no label could be found from the rules, the default is the most confident/highest supported rule (the first class in the sorted rule list). Additionally, a parameter that can be used to fine tune the classifier is k; the number of predicted classes to assign to each tuple. Since these labels are added to a set, duplicates will not show up in the set, but they are counted. This allows for a partial solution to the multiclass labeling issue unhandled by other classifiers. The statistic seen later called "online training time" refers to the total time taken to complete all of the above steps divided by the number of documents used to test the classifier.

Accuracy

Accuracy is judged by taking all of the sets of predicted labels and taking their Jaccard similarity with the actual set of labels. The reported measure is the average of all of these similarity

measures. Using Jaccard similarity allows for partial accuracy when the two sets are not disjoint, and a reported 100% accuracy when they are the same.

Results

There were 3 main parameters for the classifier: minimum support, minimum confidence, and k. Figure 1 explains what changes in minimum support do, figure 2 explains what changes in minimum confidence do, and figure 3 explains how choosing k effects the accuracy.

Support (%)	Offline Time per tuple (ms)	Online Time per tuple (ms)	Accuracy
1	1.0537	126.6539	0.5772
2	0.1189	9.2565	0.5619
3	0.0502	2.4636	0.5489
5	0.0270	0.4794	0.5226
15	0.0126	0.244	0.5067

Figure 1: This shows the effects of increasing support while keeping confidence at 40 and k at 5. A training split of 80/20 was used.

Confidence (%)	Offline Time per tuple (ms)	Online Time per tuple (ms)	Accuracy
5	0.2848	35.411	0.5475
20	0.1125	7.1064	0.5476
40	0.0496	2.5439	0.5489
60	0.0342	1.2178	0.5429
80	0.0234	1.0545	0.4381

Figure 2: This shows the effects of increasing confidence while keeping support at 3 and k at 5. A training split of 80/20 was used.

K	Accuracy
1	0.5510
3	0.5570
5	0.5489
8	0.5354
15	0.5131
25	0.4876

Figure 2: This shows the effects of increasing k while Keeping support at 3 and confidence at 40. A training split of 80/20 was used.

From the three figures on the previous page we can easily deduce the intuitive trends for each parameter. Decreasing support will consistently improve accuracy, but there is eventually a point where both the offline and online times explode, and the algorithm becomes intractable for large datasets. For this dataset a support of 2 was an optimal compromise. Increasing minimum confidence can improve speeds, but there eventually hits a point where the tradeoff for accuracy is not worth it. This appeared near 60 for this dataset. Finally, the k value had negligible change on time so it was not shown. It did modestly effect accuracy however. While sticking with a k of 1 could suffice, it was found that when k is 3 the optimal accuracy was found for this dataset. This value is highly individual to the type of dataset used and how many class labels are in the average testing transaction.

Comparing Results

As a reminder of the last lab and for purposes of comparison I will reintroduce the results from the KNN, Naïve Bayesian, and Decision Tree classifiers. Since the best training split for each of these was unanimously 66/33, the same was used for the Association Rule classification for consistency. The comparison can be seen in figure 1.

Classifier	Offline Time per tuple (ms)	Online Time per tuple (ms)	Accuracy
KNN	0.096	9.963	0.7123
Bayesian	0.0169	0.6918	0.6145
D-Tree	0.2062	0.0031	0.7041
AR	0.0628	3.8453	0.5704

Figure 1: KNN, Naïve Bayesian, Decision Tree, and Association Rule classifier results on full Reuters dataset with a training split of 66/33.

The optimal parametric constraints found in the previous section were used in the Association Rule classifier. The minimum support was 2%, the minimum confidence 60%, and k was 3. The results for classification were less than astounding nonetheless. This classifier ended up performing worst in accuracy, and second worst in efficiency. It can be noted however that the

other 3 classifiers all used an accuracy method where if the singular predicted class was in the set of actual class labels, it was counted as 100% accurate. Using this measure, the accuracy for the AR classifier improves to 62.39%, and it marginally beats the Naïve Bayesian in accuracy, but is still slower.

Conclusion

It can be concluded that in general, to optimize an Association Rule classifier, you should try to lower support and raise confidence. The AR classifier is scalable only if the support is kept high enough and the feature vectors are relatively small. The problem can quickly become intractable if the proper pruning is not conducted and constraints are not pushed into the data mining process. Overall, this dataset had far too unevenly distributed class label frequencies to be effective for AR classification. The minimum support needed to raise the accuracy to something like that of KNN or Decision Tree would cause the runtime to extremely impractical. It is still advised that of these four methods, Decision Tree classification should be used on this type of dataset.