Eric Lewantowicz
CSE 5243 Data Mining
Homework 3: Programming Assignment 2
report2.doc

For CL-1, I developed a K-Nearest Neighbor lazy learner classifier from scratch using Python 2 (see submitted project2.py source code; output files have also been submitted that show more detailed output statistics). Starting with the base the algorithm discussed in class, I developed the classifier manually. For each test sentence, its words are compared against each of the training set sentences. If a training sentence contains a matching word from the test sentence, the similarity value for the training sentence is incremented. After comparing the test sentence to every training sentence, the similarity scores are sorted, and the highest K similarity scores are chosen. The sentiments from those K similar training sentences are then examined, and majority voting is used for those sentiments to predict the sentiment of the test sentence. The actual sentiment of the test sentence is then compared to the prediction to calculate the accuracy of the classifier.

I incorporated some additional features to the K-NN classifier to attempt to improve its baseline prediction accuracy. First I randomly divided the 3,000 dataset sentences into three sets—60% training set, 20% validation set, and 20% testing set. The program first runs the classifier using the validation sentences for k = 1, 3, 5, 7. The resulting accuracy from each iteration of k is tracked, and the k that yields the highest accuracy is chosen to classify the test set sentences. Also in the first project phase, I created three separate dictionaries of the dataset words, experimenting with the NLTK-Stemming and NLTK-Lemmatization functions. So now, during the validation phase, for each k, I use each of the three dictionaries (manually parsed, NLTK Stemmed, and NLTK Lemmatized) to create the similarity scores and predictions, and the dictionary that produces the highest accuracy is then chosen to use to run the K-NN classifier on the test set sentences. The overall idea in this approach is that Stemming and Lemmatization create word roots, reducing the overall number of unique words in the dataset, therefore creating more occurrences of similar words between sentences and increasing similarity scores between sentences. It took approximately 8 seconds to parse the raw data and create all three dictionaries. (Note: All the tests were conducted remotely logged into stdlinux, and there appeared to be a variability of execution times depending on the time of day and stdlinux's workload. For example, it took 8 seconds to build the dictionaries at approximately 1:00am, but the next day at 12:00pm it took 40+ seconds to build the same dictionaries.)

Finally, I attempted to improve the algorithm by factoring negative-word correction into the classifier. For example, if a training sentence matches a test sentence with the word "good", but the training sentence also contains the word "not", "isn't", etc., but the test sentence does not contain negative descriptors, then there is some chance that the sentiment of the training sentence should be opposite of the sentiment of the test sentence, even though they appear to have a high similarity due to other matching words. So for each k, and for each dictionary, the classifier also predicts the validation sentiment both with and without negative word correction. Whichever parameter combination of k, dictionary, and negative word correction produces the highest accuracy in the validation set is then used to classify the test set.

After completion of the validation set classification with the K-NN classifier, the test set is then classified using the three optimal parameters decided from the validation set—optimal-k, optimal-dictionary, and negative word correction or not. The below output block summarizes the results from the 600-sentence validation set using the K-NN classifier on the original FV-1 vector before pruning:

Current k: 1
Time per tuple: 0.12 seconds
Best accuracy with k = 1 ; NotCorrection: False ; using NLTK Stemmed words dictionary
Accuracy: 65.50

Current k: 3
Time per tuple: 0.13 seconds
Best accuracy with k = 3 ; NotCorrection: False ; using NLTK Stemmed words dictionary
Accuracy: 67.17

The best accuracy during this instance of program execution in the validation set was 67.17%, and it was achieved when k = 3 (using the 3 most similar nearest neighbors), using the NLTK Stemmed words, and not using the negative-word correction. The time to classify a tuple before pruning ranges from 0.1 – 0.2 seconds depending on the value of k and the dictionary used. The optimal parameters were then fed into the test set sentences, and the K-NN classifier was executed again. The results are shown below:

kNN predictor Test Set results BEFORE pruning using validation set optimizations:
optimalWords Test: Stem words optimal
optimal k: 3
not-correction: False
Time per test tuple: 0.10 seconds
Correct predictions w/out NOT correction: 400
Wrong predictions w/out NOT correction: 200
Accuracy: 66.67

Running the 600 test set sentences through the K-NN classifier on FV-1 (before pruning) with k = 3 and using the NLTK Stemmed dictionary vector, we achieved 400 correct predictions and 200 wrong predictions, for an accuracy of 66.67%. It took 0.10 seconds to classify a tuple.

Running the 1,800 FV-1 training set sentences through the CL-1 classifier after the validation set identified the optimal k-value produced a slightly higher accuracy than the test set—67.72% training accuracy versus 62.67% in one sample program execution with optimal k=1. The training accuracy was determined using a leave-one out approach, where the matching training sentence was ignored from the training set, and its similarity was not factored into the K-NN calculation, so as not to positively bias the training accuracy. Time to classify was 0.02 seconds per tuple.

After completing CL-1/FV-1, the next step was feature selection and pruning of FV-1 to create FV-2. For pruning, I first deleted any single-occurrence words from the frequency vectors. That is, if a word only occurs one time throughout the dataset, it has no reference to compare against in other sentences and is likely not useful in a classifier. Pruning single-occurrence words reduced the vector unique word counts from by approximately 2,200 – 3,100 words from each of the three vectors, making the vectors smaller by over 50% compared to the original unpruned vectors.

Pruning singles:
Original word count: 5383 (manual parsed vector)
After delete word count: 2240

Original word count: 4189 (NLTK Stem vector)
After delete word count: 1952

Original word count: 4869 (NLTK Lemma vector)
After delete word count: 2136

I then pruned high-entropy words where the sentiment was evenly split between the occurrences, i.e. there were the same number of occurrences of a word in positive-sentiment sentences as there were in negative-sentiment sentences. The intuition to this pruning is that these high-entropy words would not help a classifier to decide whether a sentence was positive or negative.

The final pruning step used principles similar to those of TF-IDF, in that I used two independent threshold variables to consider both the entropy of a word, and its frequency of occurrence. The idea is to prune words that are frequently occurring throughout the frequency vector that are relatively high entropy and do not strongly classify a similar test sentence. Continuing to test different occurrence threshold and entropy thresholds is an opportunity for additional accuracy improvement in this project.

High occurrence words, low selective words deleted:
Prune occurThreshold: 100
Prune posNegThreshold: 0.4
After delete word count: 1806 (manual parsed)
After delete word count: 1580 (NLTK Stemmed)
After delete word count: 1711 (NLTK Lemmatized)
Time to prune: 80.78 seconds

After the final pruning step, the word vectors are approximately 35% of the size of the starting word vectors for each of the manual, Stemmed, and Lemmatized vectors. Total pruning time for all three word vectors took 81 seconds on stdlinux. After pruning, the program again executed CL-1 on the validation set to determine the optimal k, dictionary, and negative word correction parameters to choose for the test set sentences.

kNN predictor Validation Set results after pruning:
Current k: 1
Best accuracy with k = 1 ; NotCorrection: False ; using NLTK Stemmed words dictionary
Accuracy: 72.33

Current k: 3
Best accuracy with k = 3 ; NotCorrection: False ; using NLTK Stemmed words dictionary
Accuracy: 77.17

Current k: 5
Time per tuple: 0.0257 seconds

Again, k = 3 gave the best accuracy. This is likely due to the relative sparseness of useful common word occurrences among sentences throughout the dataset, such that there is minimal similarity among a test sentence and the training sentences beyond the top three similar sentences in the set. For larger datasets with more words per sentence, we may expect optimal k to be larger than three.

The optimal frequency vector was again the NLTK Stemmed word vector without negative-word correction, which gave us the best accuracy of 77.17% on the validation set. This is an improvement of 10% over the performance on FV-1 validation set. Time to classify each tuple with the pruned vectors was also significantly reduced, at approximately 0.026 seconds per tuple, compared to the 0.10 seconds with the unpruned tuples. Using the optimal k and feature vector parameters from the validation set to then classify the test set with the pruned word vectors (CL-1 with FV-2) produced the results for the test set below:

Time per tuple: 0.0221 seconds
Correct predictions w/out NOT correction: 446
Wrong predictions w/out NOT correction: 154
Accuracy: 74.33

After pruning and validation set optimization, we were able to achieve 74.33% accuracy in the prediction of the 600 test set sentences. Time to classify a tuple was 0.022 seconds for the test set.

As with FV-1, running the 1,800 FV-2 training set sentences through the CL-1 classifier produced a slightly higher accuracy than the test set—76.61% training accuracy versus 74.67% in one sample program execution with optimal k=9. Time to classify was 0.0085 seconds per tuple.

Overall time spent developing the K-NN classifier was more than 10 hours. This was spread over several days, and the time includes time developing and tweaking the pruning methods and optimal pruning thresholds, adjusting the dictionary creation, handling the validation set, trying different test variations, and then implementing the basic K-NN classifier and the optimal parameter selection using the validation set. Another area for improvement of this classifier is in the negative word correction. The final implementation of the negative word correction did not improve the performance, but I think this may still be a feature that can improve predictor accuracy if implemented more appropriately.

For the second classifier, CL-2, I built a binary Decision Tree using information gain, again designing and implementing it from scratch using Python 2. I first implemented a Binary Tree class, where each inner node contains the string value of a word from the dataset, and each leaf node contains a sentiment determined by majority voting of the sentiment of the sentences from the training set in the leaf node. To construct a decision tree, I again divided the dataset into 60% training, 20% validation, and 20% testing. The algorithm is as follows: using the training set, for the root node, the program iterates through each unique word attribute occurring in the set of sentences at the node. The word that produces the best information gain based on sentiments for the sentences where the word occurs or does not occur (smallest information required after using that word to split the tree) is chosen as the binary split point for the node. All sentences in the training set are then assigned to the left subtree if they do not contain the word, and to the right subtree if they do contain the word, and the word is removed from the sets of unique words for the subtrees. The algorithm is then recursively repeated on the left and right subtrees to create the tree. If all the sentences at a node are of the same class sentiment, the node becomes a leaf node and is assigned the sentiment of the sentences. Or if the sentences are mixed, but the entropy (information needed) falls below a threshold parameter, the node also becomes a leaf node and the majority sentiment value is assigned to the node. This also helps to minimize overfitting the data and creating an overly complex tree that may reduce accuracy.

Due to the relatively large size of the dataset, building a complete tree becomes time-intensive and memory-intensive. Therefore, the validation set is used to help identify the optimal depth of the decision tree and the optimal info-gain threshold parameter to stops subtree construction and assign a leaf node below the threshold information gain. Using the validation set, multiple trees are built with varying depths and info-gain threshold parameters, and the parameters that produce the best validation set prediction accuracy are saved and used as the classifier for the test set data.

For CL-2 on FV-1, the large size of the unpruned dataset made creating large decision trees time consuming. Therefore, the info gain threshold parameter was fixed to 0.20, and using the validation set, trees starting at 6 levels deep up to 21 levels were constructed at 5-level increments. The time to construct a 21-level tree with 155 nodes was 393 seconds, with a prediction accuracy of 66.17% on the validation set (compared to 59.0% on a 6-level tree with 33 nodes). I likely could have increased the CL-2 accuracy on FV-1 if I had continued to construct larger trees, but the time required to construct them would be unreasonably long. Using the 21-level tree with a 0.20 info-gain threshold parameter, the prediction accuracy achieved classifying the testing set was 68.83%, with a time to classify tuples of 0.0001 seconds per tuple. Compared to the CL-1 K-NN classifier, the vast majority of the time is spent constructing the decision tree, and once constructed, classifying tuples is a very fast process. Program output for CL-2 on FV-1 is below:

Decision tree on test set before pruning using optimal tree depth and info gain threshold:
Number tree nodes: 155
Time to classify 600 tuples in tree with 21 levels: 0.06 seconds
Time to classify per tuple: 0.0001 seconds
accuracy: 68.83
infoValThresh: 0.20

Running the 1,800 training set sentences through the optimized decision tree of FV-1 produced a training accuracy of 74.22% (versus 67.33% on the test set during this program iteration). The relatively low accuracy of the training set on the tree is likely due to the low depth of the tree and relatively small number of nodes.

The same pruning process used before was again used to create FV-2 from FV-1. Due to the longer model construction time, only the manually parsed word vector was used for CL-2. The dataset was reduced from 5,383 unique words to 1,806 words. The decreased size of FV-2 significantly reduced the time required to build larger decision trees, allowing for the construction of larger trees and more testing of different info-gain threshold parameters, resulting in greater accuracy with CL-2 for FV-2 versus FV-1.

For CL-2 on FV-2, the validation set was again used to determine the optimal decision tree size, now between 26 and 61 levels, and for info-gain threshold parameter from 0.1 to 0.7 at increments of 0.2. Constructing a 26-level tree with 141 nodes took 132 seconds, and a 61-level tree with 354 nodes took 243 seconds to construct. The best accuracy was generated by a tree with 61 levels, 354 nodes, and an info-gain threshold parameter of 0.1, producing an accuracy of 73.67% on the validation set. Using this optimal tree, the test set was then classified, at a time of 0.0002 seconds per sentence, and an accuracy of 76.33%. An additional test constructing trees up to 91 levels did not produce significantly better accuracy. Program output for CL-2 on FV-2 is below:

Decision tree classifier on test set after pruning using optimal tree depth and info gain threshold:
Number tree nodes: 354
Time to classify 600 tuples in tree with 61 levels: 0.15 seconds
Time to classify per tuple: 0.0002 seconds
accuracy: 76.33
infoValThresh: 0.10

Running the 1,800 training set sentences through the optimized decision tree of FV-2 produced a training accuracy of 81.00% (versus 76.17% on the test set during this program iteration).

Time spent to build and test the Decision Tree classifier was more than 8 hours, but much of that included various testing iterations and evaluating accuracy under different conditions after the model was built. Areas for improvement include continuing to construct larger trees with more depth, further adjusting the info-gain threshold parameter, and introducing different pruning techniques and/or weighting techniques to compensate for overfitting and noise within the training set.