COEN 281, Homework 4
Due: Tuesday, December 9
(This final homework is to be done individually)

Please turn in a paper copy in the department. Email should only be used as a ***last*** *resource (i.e., you are out of town). Work turned in d days late is graded and the grade is multiplied by (1 – d/10) if d ≤ 3, and 0 otherwise.*

1. Ensemble Methods. The *13_Diamond_Data.xls* file contains information on 6000 diamonds; read the accompanying README file for a description of the different attributes. The goal in this problem is to build a regression model to predict the wholesale market value of a unique stone.

You can use the *read.xls()* function from the *gdata* package to read the data file, or convert it to text from Excel and read it with *read.table()*. We are going to use the *RuleFit3* package for building rule-ensemble models. It is available for download from: http://statweb.stanford.edu/~jhf/R_RuleFit.html. Use RuleFit directly within R, or alternatively try the command line interface available at github.com/intuit/rego.

(a) Show the distribution of the “cut” and “Price” attributes.

(b) Use the *rulefit()* command to fit a regression model (i.e., *rfmode="regress"*) to a random sample of size 5000. Note that rulefit takes a “cat.vars” argument to identify the categorical attributes. Feel free to experiment with various values of the rulefit’s arguments, but in you final submission use *test.reps* =10 and *test.fract*=0.1. Use the *rfmodinfo*() command to report average absolute error and number of terms (the ensemble size).

(c) Use the *rules()* command to view the top 10 rules [Note: if the invocation of rules() leaves you in the command prompt, you can type *quit* to return to R]. i) Was there a linear term in the model? Explain its coefficient. ii) Explain one of the other rules.

(d) Use the *varimp()* command to variable importance plot. Which are the top three most important variables in determining a diamond’s price?

(e) Report the error on the test set using “Average Absolute Error” – i.e., $\frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$

(f) Fit a decision tree to the same training data and report the error of the best pruned tree.

2. Partition-based Clustering. Use *kmeans()* to find clusters in all the letters data. Do not include the first column. One of the values returned by this function is *withinss* - the within-cluster sum of squares for each cluster. We use the average *withinss* as a “goodness of fit” criterion for a given value of K – i.e.,

$$J(C) = \frac{1}{K}\sum_{k=1}^{K} withinss(C_k)$$

(a) Repeat kmeans for K=2,…,26, computing and saving *J(C)* in each case.

(b) Plot the goodness of fit values vs. K twice. First for all K, then for K=15,…,26 only. – do you notice any "steps" that might suggest the number of "natural" clusters?

3. *Hierarchical Clustering*. Use *hclust()* with the $d_{Average}$ distance method to form a dendogram of the 26 means from the last run of k-means. These means are given in the '*centers*' value returned by *kmeans()* -- i.e., *kmeans$centers*.

(a) Show the dendogram. There should be a cluster number (or index) on each leaf.

(b) Assigning letter labels to each center (node in the dendogram). Function *kmeans()* also returns a `*cluster*' vector with integers indicating the cluster to which each point is allocated. Build a 26x26 matrix (i.e., letters by cluster-number) to determine the most common letter associated with each given cluster. Show the dendogram above with these letters instead of the cluster numbers.

(c) Report two observations (analysis) of what the dendogram tells you (anything you find interesting).

(d) In the dendogram with letter labels you will notice that some letters are missing and some are duplicated. Determine to what cluster we should assign each missing letter.

4. *Association Rules*. The data files "movies.txt" and "ratingsAsBasket.txt" contain movie ratings from Netflix (Be sure to read the license note from that company which accompanies the data.). The goal of this problem is to learn how to use the Apriori algorithms to find association rules. Each line in "ratingsAsBasket.txt" represents all the movie ratings provided by a single user. Not all users rated the same number of movies. Movies are represented by numbers and ratings have been summarized by three values: 'Low", "Medium", and "High." Thus, a "basket" will be the set of all movies rated by one person, and the "items" are the pairs <movieId, rating indicator>.

The R function apriori from the arules package provides the apriori functionality using Borgelt's implementation.

(a) Use the *read.transactions()* command to load the data into R. Show a summary of the dataset with the command *summary()*. Using information from this summary, find out:

- Number of "baskets"
- Describe most frequent item (i.e., title, rating, frequency)
- Minimum/Maximum/Average number of movies rated by one rater

(b) Use the *apriori()* command to build association rules from the dataset (default values for the parameters are ok but feel free to experiment). Use the *inspect()* function to view the actual results of the modeling.

- Show the top-10 rules
- Substitute the movie titles for one of these top-10 rules. Comment.

(c) Explain what is “lift’ of a rule. Use the *subset()* command to list all the rules which have a lift greater that 3.0

- Substitute the movie titles for one of these rules. Comment.

5. *Text Mining*. The R package *tm* provides functionality for text mining applications. The data directories “rec.autos” and “rec.motorcycles” contain a subset of the 20 Newsgroups data set.

(a) Use the *DirSource()* and *Corpus*() tm commands to load the data into R (you may want to use the *reader = readPlain* argument in the call to *Corpus()*). Confirm all the documents were read into the corpus by

- Showing the output of *length(news.corpus)*, where news.corpus is the variable where you stored the output of *Corpus()*
- Printing the corpus entry corresponding to document rec.autos/103806 (hint: use *names(news.corpus)* and [[]] indexing to access corpus entries).

(b) Use the *tm_map()* command to apply the following preprocessing transformations of the corpus (in order): *removePunctuation*, *removeNumbers*, *tolower*, *removeWords* (with argument *stopwords("english")*) and *stripWhitespace*. After each step, print document rec.autos/103806 to confirm the effect of the operation.

After the last transformation, please use the command

```
news.corpus <- Corpus(VectorSource((news.corpus))
```

to make sure the corpus elements are left in plain text prior to the next step.

(c) Use the *DocumentTermMatrix()* command to create a document-term matrix with TFIDF weights (also use *minWordLength = 1* and *minDocFreq = 1* arguments). Then

- Use *dim()* to report the dimensions of the resulting matrix. This is a very sparse matrix with far fewer non-zero entries than the *dim()* values would suggest.
- Use the *inspect()* command to confirm that the terms 'bmw' and 'clutch' are present in our rec.autos/103806 document but 'mother' isn’t.

6. *Visualization*. The R function *image* makes an image of a matrix. You can use it to visualize the off-diagonal elements in the (test set) confusion matrix – i.e., to easily spot the most problematic letters. To this end you should clear (e.g., set to 0) the diagonal elements before invoking the command.

(a) Turn in an image showing the non-zero entries colored. Use 4 colors only and no numeric ticks or labels. Use the *axis* command to add the letter labels to the plot.

(b) Identify the color corresponding to the worst confusions and list the corresponding character pairs.