Rtexttools: A Supervised Machine Learning Package in an R Wrapper

Loren Collingwood and John Wilkerson
Center for American Politics and Public Policy
University of Washington
NOTE:Rtexttools IS NOW DEPRECATED.
PLEASE GO TO rtextools.com
FOR THE MOST CURRENT VERSION

June 8, 2011

1 Introduction

`TextTools` relies on pre-labeled document examples to train algorithms to classify other documents. This 'supervised learning' approach is widely used within computer science. Our example will be a sample of Congressional bill titles. Humans have assigned one of 19 primary topic codes to each bill title.

The `TextTools` package itself was created by Paul Wolfgang based on earlier work by Dustin Hillard and Stephen Purpura. It includes four algorithms, SVM, MaxEnt, LingPipe and Naïve Bayes, and customizable pre-processing components that are described in the Text Tools documentation: `http://www.cis.temple.edu/~wolfgang/Text%20Classification%20Tools.pdf`. The R wrapper, `Rtexttools`, that we will be using here (developed by Loren Collingwood) first prepares and sends jobs to `TextTools` for processing. Then we import the results into R to take advantage of R's statistical offerings more generally.

2 Steps in the Process

- Create a training database
- Train the algorithm
- Test the algorithm against labeled documents
- Refine through Active Learning
- Classify unlabeled documents

Import/Create a training database. A set of pre-labeled documents is required. It must be large enough so that part of it can be used to train the system, while another part is used to evaluate system performance. There is no simple answer for how many labeled training cases are required (other than more are better!), but a rough starting point might be 100 examples per category.

Train and Classify. Training the algorithms involve several steps. The general goal is to distinguish the relevant features of documents that leads humans to assign them to different classes. A central challenge is that documents can contain irrelevant similarities (or differences) that lead to classification errors. **Pre-processing** helps to mitigate such errors by word 'stemming' (e.g. eliminating suffixes such as 'ing') and by removing common 'stopwords' (e.g. a, the, or). The next step is to train the algorithm(s). Then the trained algorithms are used to classify new documents.

Test the algorithm against labeled documents. It is important to know how well the system is working. This is typically done by predicting the labels (classifying) of another set of pre-labeled documents. Then the machine results can be compared to the pre-existing labels. Text Tools includes options to assess the overall performance of an algorithm, precision, recall and bias (via a confusion matrix), and algorithm agreement. Typically, information acquired at this stage is used to improve performance through active learning.

Active Learn. Active learning generally refers to using information obtained during testing to improve the training process. Which adjustments can improve performance most efficiently? For example, we could increase the size of the training set by randomly classifying additional documents. But if the errors are not randomly distributed, it might be more efficient to discover and make adjustments in those areas where the system is performing especially poorly. The **confusion matrix**, which compares actual and predicted labels, is well suited for this purpose.

Classify unlabeled documents. The active learning process continues until the researcher is satisfied with the system's performance. If overall performance is unacceptable, an alternative approach is to separate the cases that do meet a performance threshold. Multiple algorithms can sometimes be used as a measure of confidence in a label. If experiments indicate high reliability when multiple algorithms predict the same label for a case, then cases where the **ensemble** agrees can be set aside, substantially reducing the cases that require human attention.

Performance Assessment. How do we know how well the algorithms are working? The common baseline is to ask whether the machine labels as reliably as the typical human, by comparing human-human agreement with machine-human agreement. However, both humans and machines should do better than what we would expect by chance.

A variety of statistics are available that compare observed agreement with expected agreement. We will use **Krippendorf's Alpha** in this example. The confusion matrix provides additional information about agreement with respect to particular labels, as well as information about the distribution of errors (bias).

We might also want to run experiments using different training sets, and by alternating the portion of the test data that is used to test, to test the sensitivity of our results. For some projects, it may also be possible to assess reliability using other indicators (for example, if another project has classifies bill topics based on media coverage rather than titles).

3 Setting up the R Package

Download and install `R` on your desktop. For information about how to do this:

`http://faculty.washington.edu/jwilker/559/PAP/install_R.pdf`

Download the `Rtexttools` zip file to your desktop, or wherever your browser downloads to. The latest version can be found here:

http://staff.washington.edu/lorenc2/software/index.html

Open `R`. You may need to install some additional packages. If so, this is how you install a package:

```
> install.packages("RODBC")
> install.packages("concord")
> install.packages("micEcon")
```

Then load them via the `library` function.

```
> library(RODBC)
> library(concord)
> library(micEcon)
```

At the top menu, go to:

- Packages
- Install packages from local zip files
- Then navigate to where you downloaded the `Rtexttools` zip file.

The package will install in a subdirectory of `R` on your computer. (You may have to close and reopen `R`) . This package may also take longer than normal packages to install because it is large. However, it shouldn't take longer than about 5-7 minutes. When it's finished, in the `R` command line, type:

```
> library(Rtexttools)
```

To access the documentation/help files, type:

```
> `?`(Rtexttools)
```

Take a quick look through the documentation and note the 'index' at the bottom.

4 Import a Train-Test Access database and ODBC register

We are going to use an example database that is already part of the R package ("Spanish.accdb"). This is a smallish file of Spanish legislation and U.S. Congress bills that have been manually coded for topic.

If you wanted to import your own (Access) database, it might be easiest to put it in the same working directory. It must be an Access database which is ODBC registered on the computer you are currently using.

Search 'ODBC' on your computer. When the program opens, click the 'Add' option; select Microsoft Access Driver (.mdb, accdb) option; click Finish; Find and select the 'Spanish.accdb' file; Click OK; Name the Data Source = spanish (could be anything but this is what it is called in the example code below); click OK.

If you are using Windows 7 and the Access Driver is not listed, please go here for additional information:

http://Faculty.washington.edu/jwilker/559/ODBCWindows7.pdf

Now, so that you will appreciate what we are about to do...

Open the 'Spanish.accdb' database. It will be in your R directory under

~/library/Rtexttools/TextTools_V0.11/Spanish.accdb.

There are 4 tables in Spanish.accdb. We will use the labeled titles in the Congress_Train table to train the algorithms, and the titles in the Congress_Test\verb to test them. Note that both tables include titles and human assigned labels ("major") but that only the second table has columns to accept the algorithm's predictions (new_code_svm, etc).

5 Train and Test using labeled documents

To run an example of any of the Rtexttools functions found in the documentation (e.g. texttrain; textclassify), simply type: example(functionname). However, we will do this as if we were working on our own project. First, we need to make sure that we are in the directory where the Access database is located:

```
> getwd()
```

```
[1] "C:/Users/Loren/Documents/CAPPP/texttools/Rtexttools/inst/doc"
```

```
> setwd(system.file("TextTools_V0.11", package = "Rtexttools"))
```

Next, we specify the data for training and the algorithm to be trained. We are only going to do this for one (SVM) here. The documentation has example codes for the other algorithms. Copy and paste this to the `R` command line and run it.

Look at the arguments

```
> args(texttrain)
```

```
function (method, datasource, table_name, id_column, text_column,
    code_column, model, feature_dir = "SVM_Training_Features",
    use_even = "false", use_odd = "false", compute_major = "false",
    preserve_case = "false", remove_stopwords = "true", do_stemming = "true",
    trainer = "NaiveBayes", language_model = "N-Gram", gram_size = "6",
    categories = "1 2 3 4 5 6 7 8 10 12 13 14 15 16 17 18 19 20 21 24 99",
    parameters = "Xmx1G", ...)
NULL
```

Look at the help file/documentation

```
> `?`(texttrain)
```

Now create the training objects for each algorithm.

```
> tsvm <- texttrain(method = "svm", datasource = "spanish", table_name = "Congress_Train",
+     id_column = "obs", text_column = "text", code_column = "major",
+     model = "svm_congress")
> tling <- texttrain(method = "LingPipe", datasource = "spanish",
+     table_name = "Congress_Train", id_column = "obs", text_column = "text",
+     code_column = "major", language_model = "PorterStemmer",
+     model = "ling_congress")
> tnaive <- texttrain(method = "MALLET", datasource = "spanish",
+     table_name = "Congress_Train", id_column = "obs", text_column = "text",
+     code_column = "major", model = "naive_congress", trainer = "NaiveBayes")
> tmaxent <- texttrain(method = "MALLET", datasource = "spanish",
+     table_name = "Congress_Train", id_column = "obs", text_column = "text",
+     code_column = "major", model = "maxent_congress", trainer = "MaxEnt")
```

Create the classification objects for each algorithm.

```
> csvm <- textclass(method = "SVM", datasource = "spanish", table_name = "Congress_Test",
+     id_column = "obs", text_column = "text", code_column = "major",
+     output_code_col = "new_code_svm", model = "svm_congress")
> cling <- textclass(method = "LingPipe", datasource = "spanish",
```

```
+     table_name = "Congress_Test", id_column = "obs", text_column = "text",
+     code_column = "major", output_code_col = "new_code_ling",
+     model = "ling_congress")
> cnaive <- textclass(method = "MALLET", datasource = "spanish",
+     table_name = "Congress_Test", id_column = "obs", text_column = "text",
+     code_column = "major", output_code_col = "new_code_naive",
+     model = "naive_congress")
> cmaxent <- textclass(method = "MALLET", datasource = "spanish",
+     table_name = "Congress_Test", id_column = "obs", text_column = "text",
+     code_column = "major", output_code_col = "new_code_maxent",
+     model = "maxent_congress")
```

Use 'classrun' to send each algorithm to `TextTools`.

```
> classrun(tsvm, csvm, ttpath = system.file("TextTools_V0.11",
+     package = "Rtexttools"))
> classrun(tling, cling, ttpath = system.file("TextTools_V0.11",
+     package = "Rtexttools"))
> classrun(tnaive, cnaive, ttpath = system.file("TextTools_V0.11",
+     package = "Rtexttools"))
> classrun(tmaxent, cmaxent, ttpath = system.file("TextTools_V0.11",
+     package = "Rtexttools"))
```

The classrun command may take a while to run. When the job is finished, return to "Spanish.accdb", open it, and look at the `Congress_Test` table. There should now be values in the column to the right of the titles (`new_code_svm`)?

6 What have we done?

If you look at the `R` code above, a lot of it should be familiar. For instance, the first method is SVM. The "datasource" is the OBDC name you gave for "Spanish.accdb" (spanish). The `texttrain` "table" is the first table in Spanish.accdb (the `textclassify` table is second table in the database). The "text" is located in the "text" column of the respective tables, while the labels (codes) are found in the "Code" column.

7 Performance Assessment and Diagnostics

Now to run some diagnostics.

7.1 Confusion Matrix and Diagnosing the Trained Data

The confusion matrix offers diagnostic information that may lead to more efficient strategies for improvement. The table should like something like this, albeit with different labels.

	Health	Energy	Environment
Health	14	1	1
Energy	0	28	2
Environment	13	2	6

Table 1: Generic Confusion Matrix

The rows are the human assigned labels and the columns are the predicted labels. The diagonals are the cases of agreement while the off diagonals are the prediction errors. The rows tell us how many 'true' cases the system correctly recalled. For example in the second row, we learn that the system correctly labeled 28 of the 30 human labeled energy cases. The columns tell us how many cases predicted to be about energy are about energy ("precision"). In this contrived example, 28 of the 31 cases labeled as energy were actually about energy.

Overall reliability is 72% (48/67). However, the value of the confusion matrix is that it also indicates that the system is doing better for some topics than for others. Environment has low recall (6/21), and better precision (6/9). Health, on the other hand, has higher recall but lower precision.

Knowing this can lead to more efficient approaches to improving overall performance. For example, if health and environment bills share common – but irrelevant – words, then we could expand our list of stopwords to include them. Another possibility is that some of the labels assigned by humans are incorrect. Fixing those human errors might also improve system performance.

To generate a confusion matrix, we must access the data frame. We use the `datagrab` command to bring in the `Congress_Test` table to evaluate the accuracy and reliability of the algorithms.

```
> uscongress <- datagrab("spanish.accdb", "Congress_Test")
```

Look at the data

```
> head(uscongress)
```

```
  ID    obs cong billnum h_or_sen major minor
1  1 388986  107    4499       HR    18  1807
2  2 388987  107    4500       HR    18  1807
3  3 388988  107    4501       HR    18  1807
4  4 388989  107    4502       HR    18  1807
5  5 388990  107    4503       HR     5   530
6  6 388991  107    4504       HR    21  2102
```

```
1
2
3
4
5
6 To amend title 38, United States Code, to extend the eligibility for housing loans guaranteed b
  new_code_svm new_code_ling new_code_naive new_code_maxent
1           18            18             18              18
2           18            18             18              18
3           18            18             18              18
4           18            18             18              18
5            5             5              5               5
6           14            14             16              14
```

The `alg_reliable` command produces a confusion matrix, as well as Krippendorf's Alpha measure of reliability, percent correctly predicted, and a category percent matrix. The confusion matrix can be exported to a .csv file in the working directory by setting the filename to a character string, such as filename=`"max_conf.csv"`. Below, a graph of "truth" versus "predicted" is plotted. We do not present the confusion table, Krippendorf's Alpha, or percent predicted correctly here on account of space limitations. For demonstration purposes, we use the built in "congress" dataset.

```
> data(congress)
> conf_svm <- alg_reliable(congress, c("major", "new_code_svm"),
+     method = "nominal", pct.correct = TRUE)
```

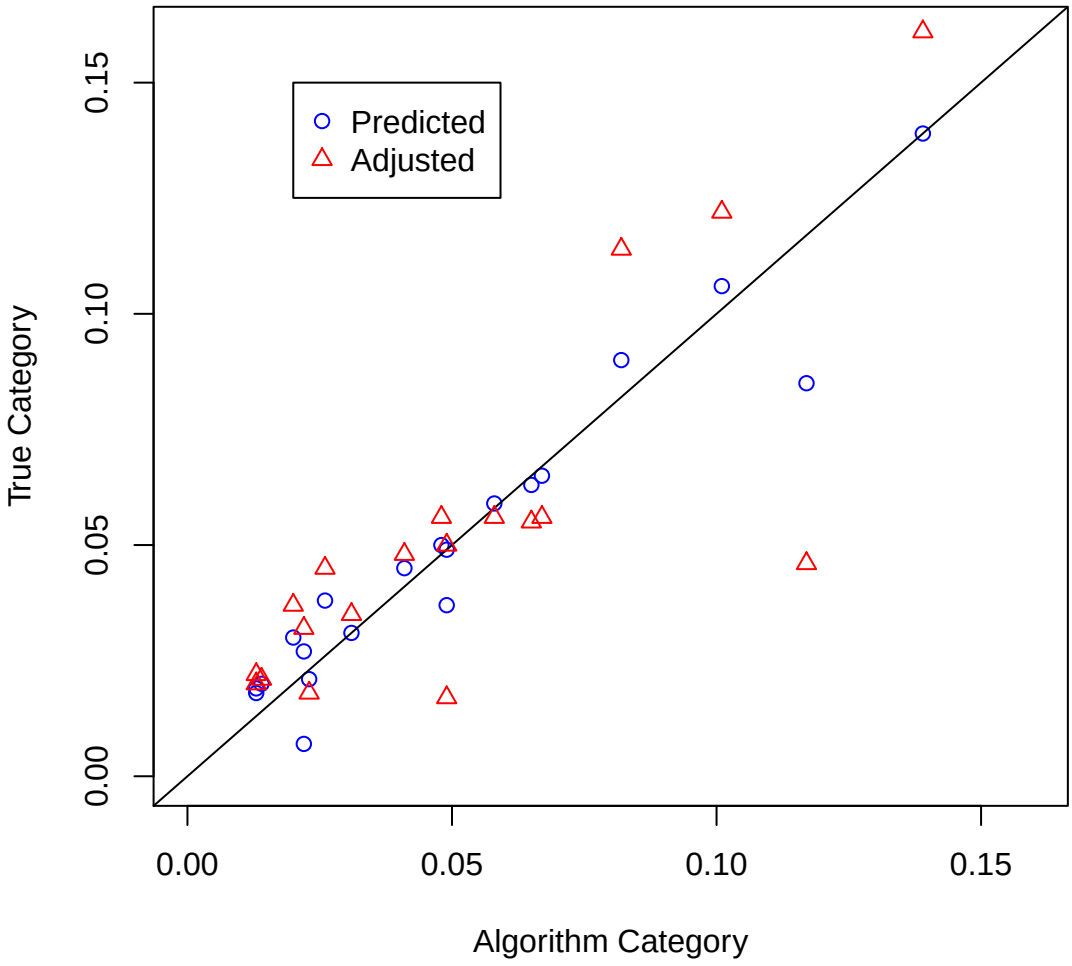

Figure 1: Truth Versus Predicted and Adjusted Plot

Now run the `alg_reliable` command for all the algorithms to evaluate their accuracy.

```
> conf_ling <- alg_reliable(congress, c("major", "new_code_ling"),
+     method = "nominal", pct.correct = TRUE)
> conf_maxent <- alg_reliable(congress, c("major", "new_code_maxent"),
+     method = "nominal", pct.correct = TRUE)
> conf_naive <- alg_reliable(congress, c("major", "new_code_naive"),
+     method = "nominal", pct.correct = TRUE)
```

7.2 Algorithm Ensemble Agreement, for Virgin Text

Confusion tables are helpful during the training process, but they are of little value once we predict to unlabeled cases (our ultimate goal). At this stage we can simply assume performance equivalent to what was learned through earlier experiments using labeled data. We can also do spot checking using random samples of the unlabeled data.

A third option is ensemble agreement. Algorithm agreement typically equates with higher precision. This information can be used to distinguish the cases that are most likely to be incorrectly labeled.

We first create a data frame that includes the classification predictions for the 4 algorithms. This is done via the `textclassify` function.

```
> install.packages("gmodels")
```

```
package 'gmodels' successfully unpacked and MD5 sums checked

The downloaded packages are in
        C:\Users\Loren\AppData\Local\Temp\RtmplS179H\downloaded_packages
```

```
> library(gmodels)
```

```
> congress_agree <- textclassify(uscongress, c("new_code_naive",
+     "new_code_maxent", "new_code_ling", "new_code_svm"))
> CrossTable(congress_agree$scores)
```

```
   Cell Contents
|-------------------------|
|                       N |
|         N / Table Total |
|-------------------------|
```

```
Total Observations in Table:  4450


          |         0 |         1 |         2 |         3 |         4 |
          |-----------|-----------|-----------|-----------|-----------|
          |        55 |       438 |       220 |      1150 |      2587 |
          |     0.012 |     0.098 |     0.049 |     0.258 |     0.581 |
          |-----------|-----------|-----------|-----------|-----------|
```

The items that score 0 and 1 can be exported for active learning coding via this coding scheme.

```
> low_scores <- congress_agree[congress_agree$scores < 2, ]
> CrossTable(low_scores$scores)


   Cell Contents
|-------------------------|
|                       N |
|         N / Table Total |
|-------------------------|


Total Observations in Table:  493


          |         0 |         1 |
          |-----------|-----------|
          |        55 |       438 |
          |     0.112 |     0.888 |
          |-----------|-----------|


> write.csv(low_scores, "low_scores.csv")
```

This file, `low_scores.csv` will be saved in the working directory. Finally, it may also be useful to write elements of the dataframe back into `Access` using the `sqlSave` function from the `RODBC` package. In the following example, we create a dataframe, `high_scores`, and write this to the Access connection entitled channel. This creates a table in Access called `high_scores`.

7.3 FindClusters

This diagnostic tool can be used to isolate cases with similar text but inconsistent labels (codes). For example, it can be used to check that human coders have labeled similar cases consistently before proceeding to the next step of training the algorithms. Or it could also be used to discover similar cases that algorithm is labeling inconsistently.

The default approach generates a report that includes only those clusters with inconsistent labels. The `code_column` specifies the labels that interest you. Here we are using the human labeled examples. The `cluster_column` doesn't appear to affect the output. We are using the table that includes both human and machine labels so we specify it to be the SVM machine predictions (`new_code_svm`). In the output below, inconsistent human labels (Code) are the focus, but reversing the specification would report only those clusters where the machine labels (`new_code_svm`) were inconsistent.

As mentioned, FindClusters only reports the clusters with inconsistent labels. If for some reason you wanted to examine every cluster, then you just need to specify a `code_column` variable that is unique - such as the id column "obs." This would then report all clusters (because all of the "codes" would be unique and therefore inconsistent).

```
> findclusters(datasource = "spanish", table_name = "Congress_Test",
+     id_column = "obs", text_column = "text", code_column = "major",
+     cluster_column = "new_code_svm", output_file = "clustertest.doc",
+     findclusters = "FindClusters", parameters = "Xmx1G", run = TRUE,
+     ttpath = system.file("TextTools_V0.11", package = "Rtexttools"))
```

Bill	Title	Code	Cluster
393344.0	A bill to provide for the testing of chronic wasting disease and other infectious disease in deer and elk herds, to establish the Interagency Task Force on Epizootic Hemorrhagic Disease, and for other purposes.	4	3
390095.0	A bill to To provide for the testing of chronic wasting disease and other infectious disease in deer and elk herds, to establish the Interagency Task Force on Epizootic Hemorrhagic Disease, and for other purposes.	7	3

Table 2: Findclusters example. See the Word file in the working directory.

8 Unadulterated R Code

```
#########################################################################
#Tools as Text Conference, Seattle, Washington                          #
#Rtexttools Example Run through                                         #
#Loren Collingwood, Center for American Politics and Public Policy      #
#6/14/2010 - 6/15/2010                                                  #
#########################################################################

#Libraries
library(concord)
library(micEcon)
library(RODBC)
library(Rtexttools)

install.packages("yhat") #Random Package for Demonstration Purposes
#Examine the documentation files, set to http. Click on index
?Rtexttools

#Working Directory for later; set to the working directory of your job
#Note, this will change for you
#texttools <- "F:\\texttools\\texttools_example"

#Set Working Directory to where the TextTools program is actually located
setwd(system.file("TextTools_V0.11", package="Rtexttools"))

#View the directory; can use this when you ODBC your database
#Also consider opening "Text Classification Tools.pdf"
getwd()

#Create Train objects that will be sent to TextTools

#Look at the arguments
args(texttrain)
#Look at the help file/documentation
?texttrain

tsvm <- texttrain(method="svm", datasource="spanish", table_name="Congress_Train",
    id_column="obs",
        text_column="text", code_column="major", model="svm_congress")

tling <- texttrain(method="LingPipe", datasource="spanish",
    table_name="Congress_Train", id_column="obs",
         text_column="text", code_column="major", language_model="PorterStemmer",
             model="ling_congress")

tling <- texttrain(method="LingPipe", datasource="spanish",
    table_name="Congress_Train", id_column="obs",
         text_column="text", code_column="major", language_model="PorterStemmer",
             model="ling_congress",
         categories= paste(1:50,collapse=" "))


tnaive <- texttrain(method="MALLET", datasource="spanish", table_name="Congress_Train",
    id_column="obs",
          text_column="text", code_column="major",
               model="naive_congress",trainer="NaiveBayes")

tmaxent <- texttrain(method="MALLET", datasource="spanish",
    table_name="Congress_Train", id_column="obs",
           text_column="text", code_column="major",
                model="maxent_congress",trainer="MaxEnt")

#Create Classify/Test objects that will be sent to TextTools

csvm <- textclass(method="SVM", datasource="spanish", table_name="Congress_Test",
    id_column="obs",
                  text_column="text", code_column="major",
                       output_code_col="new_code_svm", model="svm_congress")

cling <- textclass(method="LingPipe", datasource="spanish", table_name="Congress_Test",
           id_column="obs", text_column="text", code_column="major",
```

```
            output_code_col="new_code_ling",model="ling_congress")

cnaive <- textclass(method="MALLET", datasource="spanish", table_name="Congress_Test",
             id_column="obs", text_column="text", code_column="major",
                 output_code_col="new_code_naive",
             model="naive_congress")

cmaxent <- textclass(method="MALLET", datasource="spanish", table_name="Congress_Test",
              id_column="obs", text_column="text", code_column="major",
                  output_code_col="new_code_maxent",
              model="maxent_congress")

#Use the classrun example to send both objects to TextTools
classrun(tsvm, csvm,ttpath=system.file("TextTools_V0.11", package="Rtexttools"))
classrun(tling, cling, ttpath=system.file("TextTools_V0.11", package="Rtexttools"))
classrun(tnaive,cnaive,ttpath=system.file("TextTools_V0.11", package="Rtexttools"))
classrun(tmaxent,cmaxent,ttpath=system.file("TextTools_V0.11", package="Rtexttools"))

#When these commands are done executing, look at your Congress_Train table in your
    Access Database

#A sense of how long the algorithms take.

#SVM on LC Computer
#user  system elapsed
#0.02    0.05  141.18

#LingPipe
#user  system elapsed
#0.02    0.00   62.79

#Naive Bayes
#user  system elapsed
#0.00    0.02   51.42

#Max Ent
#user  system elapsed
#0.01    0.00   57.35

#use datagrab function to create connection and bring in Congress_Train table as a
    dataframe
uscongress <- datagrab("spanish.accdb","Congress_Test")
data(congress)
#--------------------Diagnostics-------------------------#

#Confusion Matrix for evaluating your data
#Note do not want the _agree data frame.
conf_svm <- alg_reliable(congress,c("major", "new_code_svm"), method="nominal",
    pct.correct=TRUE, filename="max_conf.csv")

#Plot Predicted and Adjusted against "True"
#Put matrix into data frame to plot vectors
   #note to LC, may want to put in category names somehow
svm_conf_data <- data.frame(conf_svm$category.matrix)
plot(svm_conf_data$Pred_Pct,svm_conf_data$True_Pct,  col="blue",
     ylim=c(0,.16), xlim=c(0.,.16),
     xlab = "Algorithm Category",
     ylab = "True Category",
     pch=1
     )
abline(a=0,b=1)
points(svm_conf_data$Pred_Pct,svm_conf_data$Adjust_Pct, col="red", pch=2)
legend(x=.02,y=.15, legend=c("Predicted", "Adjusted"), pch=c(1,2),
       col=c("blue","red"))

conf_ling <- alg_reliable(congress,c("major", "new_code_ling"), method="nominal",
    pct.correct=TRUE)
conf_maxent <- alg_reliable(congress,c("major", "new_code_maxent"), method="nominal",
    pct.correct=TRUE)
conf_naive <- alg_reliable(congress,c("major", "new_code_naive"), method="nominal",
    pct.correct=TRUE)

#Look at Algorithm Agreement
```

```
congress_agree <- textclassify(uscongress,c("new_code_naive","new_code_maxent",
    "new_code_ling", "new_code_svm"))
library(gmodels) #may need to install
CrossTable(congress_agree$scores)

#---------------------- Output ---------------------------#

#Active Learn
#Take 4's, put 0's and 1's back into the train dataset
low_scores <- congress_agree[congress_agree$scores<2,]
#Verify
CrossTable(low_scores$scores)

#Set to your directory
#setwd(texttools)
#Write these to a .csv file for coders
write.csv(low_scores,"low_scores.csv")

#High Scores
high_scores <- congress_agree[congress_agree$scores>1,]

#Write to Access database using RODBC package

#channel <- odbcConnectAccess2007("bill")
sqlSave(channel=channel, dat=high_scores, tablename="high_scores",rownames=FALSE,
    fast=FALSE)

#---------------------Find Clusters-------------------------#
findclusters(datasource="spanish",table_name="Congress_Test", id_column="obs",
    text_column="text",
    code_column="major", cluster_column="new_code_svm", output_file="clustertest.doc",
        findclusters = "FindClusters",
    parameters = "Xmx1G", run = TRUE, ttpath = ".")

#---------------------END OF FILE ---------------------------#
```