Checkpoint 3
April 2017

Task1 Canonicizers (Text Filter).
When we first read in the emails, we did not care about things like white space or even letter case. However, we did strip the punctuation in the Sentence class (or at least we should have… if you did not, do this now). For different types of analysis, we will want to apply different types of text filters to clean up the text. Here, we are going to build a TextFilter (textFilter.py). Note that the functions in this class are going to called **AFTER** all the Documents have been read in. The TextFilter class uses a style of object-oriented programing called Factory Generation. In this style, you provide the constructor a list of parameters of what type of version of the class you want. Here, we have two parameters. The first parameter is called filterList. FilterList is a python List that contains strings that represent the available filters. The available filters are: normalizeWhiteSpace (all 1 space between words), normalizeCase (turns all letters to lower case), stripNull (removes all characters that are not numbers or letters), stripNumbers (removes all numbers – numeric, not words), stripFiles (removes all words that are in a file in the wordfile/stripfile.txt file). You may shorten the string names of these filters, but make sure that they make sense. The second parameter is the Document that we will want to apply all these filter to. After, the user can call apply and all the filters will be applied to the Document **IN THE ORDER THEY ARE IN THE FILTERLIST**. You are required to fill all the function in this class. I would recommend using the setitem and getitem in the Document class for easy access to the Sentence classes that you will be editing.

Task 2. Text Filter User Interface.
Add the needed code to have text filter used on the Document List of both the training and evaluation Documents.

Task 3. skTree.
In Lab 10 (Animal Game), you design a very simple decision tree. A decision tree is a tool to make a decision on classifying an outcome. In the lab, we classify which animal. Here our goal is to classify a person (i.e., which person sent the email). There are normally two steps to making a decision. The first is called training. This is where we learn about the thing we want to make a decision about. This is done in the lab by asking you a bunch a questions to help build a tree. Here, we will use the emails in the train folder to build the trees. After we build the build, the second step is the classification. We will use the emails in the eval folder to do this.

[Please see in class notes about this! This can be very confusing. Ask many questions!
Also see treedemo.py for more information]

In this task, we are going to be using a decision tree already in sklearn package in Python. This will allow us to first focus on getting the data into the correct format before we have to build our own decision tree in the next checkpoint. There are 5 big parts to this.

1. We need to transform the data into the correct form that the package needs. We will want to train on the the following parameters: date, to whom was it sent, forward, reply, number of words in the message, existence of the top 10 words in the message, and existence of the bottom 10 words in the message. However, skLearn can only train on numeric information. Therefore, we will need to map these to a numeric value. We will want to change the date to the Julian day (nice one number for month/day/year), a number to represent to whom, and the rest to 0 for does not exist and 1 for does exist. Since we are always going to use the same list of parameters, it may help to add a function in Document that takes in a list of top 10, bottom 10 words, map for to whom it is sent, and it would return a list of all these parameters for that particular document.
2. [Putting the Training Documents together] Now, you will make a huge 2D python List of each of the above list as rows in the 2DList. You will also need a 1D List with the sender's email for the training step (note the sender's email address must also be mapped to a numeric value) .
3. Now we will use the train function in the SKTree class (skTree.py). The xData is the big 2D List from part 2 and yData is the 1DList. See treedemo.py for details on how to train (also called fit) the tree.
4. [Putting the Eval Documents together] Now, you will make a huge 2D python List of rows for from the Eval Documents. The 1DList will be all zeros, that you will be filled in by the prediction (eval).
5. Now we will use the eval function in the SKTree class to predict (skTree predict function call) the email sender.

All the putting together can be done in a function called "def predictSKTree(info)" in main. You can add an option in topMenu for this function call and have it print out who the messages age from!