Katie explained in a video a problem that arose in preparing Chris and Sara's email for the author identification project; it had to do with a feature that was a little too powerful (effectively acting like a signature, which gives an arguably unfair advantage to an algorithm). You'll work through that discovery process here.

A classic way to overfit an algorithm is by using lots of features and not a lot of training data. You can find the starter code in feature_selection/find_signature.py. Get a decision tree up and training on the training data, and print out the accuracy. How many training points are there, according to the starter code?

Special Note: Depending on when you downloaded the code provided for find_signature.py, you may need to change the code in lines 9-10 to be words_file = "../text_learning/your_word_data.pkl" authors_file = "../text_learning/your_email_authors.pkl" so that the files created from running vectorize_text.py are reflected properly.

In addition, if you are having trouble getting the code to run due to memory issues, then if you are on version 0.16.x of scikit-learn, you can remove the .toarray() function from the line where features_train is created to save on memory - the decision tree classifier can, in that version take as input a sparse array instead of only dense arrays.

tofix: MAKE ALL THE ARRAYS STOP BEING FILLLED WITH ZEROS

In [8]:
```
#!/usr/bin/python

import pickle
import numpy
numpy.random.seed(42)


### The words (features) and authors (labels), already largely processed.
### These files should have been created from the previous (Lesson 10)
### mini-project.
words_file = "../text_learning/your_word_data.pkl"
authors_file = "../text_learning/your_email_authors.pkl"
word_data = pickle.load( open(words_file, "r"))
authors = pickle.load( open(authors_file, "r") )



### test_size is the percentage of events assigned to the test set (the
### remainder go into training)
### feature matrices changed to dense representations for compatibility with
### classifier functions in versions 0.15.2 and earlier
from sklearn import cross_validation
features_train, features_test, labels_train, labels_test = cross_validation.train

from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(sublinear_tf=True, max_df=0.5,
                             stop_words='english')
features_train = vectorizer.fit_transform(features_train)
features_test  = vectorizer.transform(features_test).toarray()


### a classic way to overfit is to use a small number
### of data points and a large number of features;
### train on only 150 events to put ourselves in this regime
features_train = features_train[:150].toarray()
labels_train   = labels_train[:150]



### your code goes here
from sklearn.tree import DecisionTreeClassifier as dtc
clf = dtc()
clf.fit(features_train,labels_train)
pred = clf.predict(features_test)
acc = clf.score(features_test,labels_test)
print("Accuracy: ",clf.score(features_test,labels_test))
print("Training points: ",len(features_train))
```

```
('Accuracy: ', 1.0)
('Training points: ', 150)
```

Take your (overfit) decision tree and use the feature_importances_ attribute to get a list of the

relative importance of all the features being used. We suggest iterating through this list (it's long, since this is text data) and only printing out the feature importance if it's above some threshold (say, 0.2--remember, if all words were equally important, each one would give an importance of far less than 0.01). What's the importance of the most important feature? What is the number of this feature?

EVERYTHING IS STILL ZEROS

In [9]:
```
clf.feature_importances_
```

Out[9]:
```
array([0., 0., 0., ..., 0., 0., 0.])
```

This word seems like an outlier in a certain sense, so let's remove it and refit. Go back to text_learning/vectorize_text.py, and remove this word from the emails using the same method you used to remove "sara", "chris", etc. Rerun vectorize_text.py, and once that finishes, rerun find_signature.py. Any other outliers pop up? What word is it? Seem like a signature-type word? (Define an outlier as a feature with importance >0.2, as before).

gotta fix the EVERYTHING IS ZEROS

In [ ]:
```

```

Update vectorize_test.py one more time, and rerun. Then run find_signature.py again. Any other important features (importance>0.2) arise? How many? Do any of them look like "signature words", or are they more "email content" words, that look like they legitimately come from the text of the messages?

In [ ]:
```

```

What's the accuracy of the decision tree now? We've removed two "signature words", so it will be more difficult for the algorithm to fit to our limited training set without overfitting. Remember, the whole point was to see if we could get the algorithm to overfit--a sensible result is one where the accuracy isn't that great!

In [ ]:
```

```