In [2]:
```
#!/usr/bin/python

"""
    This is the code to accompany the Lesson 3 (decision tree) mini-project.

    Use a Decision Tree to identify emails from the Enron corpus by author:
    Sara has label 0
    Chris has label 1
"""

import sys
from time import time
sys.path.append("../tools/")
from email_preprocess import preprocess


### features_train and features_test are the features for the training
### and testing datasets, respectively
### labels_train and labels_test are the corresponding item labels
features_train, features_test, labels_train, labels_test = preprocess()
```

```
C:\Users\Andrew\Anaconda3\envs\conda2\lib\site-packages\sklearn\cross_validatio
n.py:41: DeprecationWarning: This module was deprecated in version 0.18 in favo
r of the model_selection module into which all the refactored classes and funct
ions are moved. Also note that the interface of the new CV iterators are differ
ent from that of this module. This module will be removed in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)

no. of Chris training emails: 7936
no. of Sara training emails: 7884
```

In this project, we will again try to identify the authors in a body of emails, this time using a decision tree. The starter code is in decision_tree/dt_author_id.py.

Using the starter code in decision_tree/dt_author_id.py, get a decision tree up and running as a classifier, setting min_samples_split=40. It will probably take a while to train.

What's the accuracy?

In [3]:
```
from sklearn.tree import DecisionTreeClassifier as dtc
clf = dtc(min_samples_split=40)
clf.fit(features_train,labels_train)
pred = clf.predict(features_test)

from sklearn.metrics import accuracy_score as sco
acc = sco(labels_test,pred)
print(acc)
```

```
0.9772468714448237
```

You found in the SVM mini-project that the parameter tune can significantly speed up the training time of a machine learning algorithm. A general rule is that the parameters can tune the complexity

of the algorithm, with more complex algorithms generally running more slowly.

Another way to control the complexity of an algorithm is via the number of features that you use in training/testing. The more features the algorithm has available, the more potential there is for a complex fit. We will explore this in detail in the "Feature Selection" lesson, but you'll get a sneak preview now.

What's the number of features in your data?

(Hint: the data is organized into a numpy array where the number of rows is the number of data points and the number of columns is the number of features; so to extract this number, use a line of code like len(features_train[0]).)

In [5]:
```
len(features_train[0])
```

Out[5]: 3785

go into ../tools/email_preprocess.py, and find the line of code that looks like this:

selector = SelectPercentile(f_classif, percentile=10)

Change percentile from 10 to 1, and rerun dt_author_id.py.

What's the number of features now?

379

What do you think SelectPercentile is doing? **Would a large value for percentile lead to a more complex or less complex decision tree, all other things being equal?** Note the difference in training time depending on the number of features.

More complex

What's the accuracy of your decision tree when you use only 1% of your available features (i.e. percentile=1)?

.967

In [ ]:
```

```