tinyurl for this etherpad: http://tinyurl.com/DST4L1029

Please vote in Chris's Doodle poll for best hack days in January.

Notebook/data for today: https://dl.dropboxusercontent.com/u/75194/d4.zip ("NaiveBayesAndFriends")

"Fit": How well a line that you can draw goes through a data set to either separate different data or unify a pattern.
"Overfitting": You don't want the model to represent the data exactly (then it isn't a model anymore).

Linear regression: drawing the line through the data
Logistic regression: drawing the line to seperate the data

we've done fits like this: **logist.fit(X,y)**, working on an array. Working on an array is standard for regression.

(Go over "Classification Boundary Plot" section of notebook on your own for a more thorough overview of scikit-learn's capabilities.)

It can't draw anything other than lines with the way that we've set up the variables.

scikit-learn can create training & test sets from your data:
from sklearn.cross_validation import train_test_split
xtrain, xtest, ytrain, ytest = train_test_split(X, y)

by default, this results in 75% of the sample going into the training set; the rest goes into the test set

every regression has a score *logisit.score()* it takes the predicted values and evaluates them
If your training set accuracy is high and your test set accuracy is much lower, your model is overfitted.

There are limits to how well you can predict anything.

Your sample needs to be balanced - if you have 9 males and 1 female in your training data, you won't be able to accurately predict. You have to do cross-validation to iterate through different sets of training/test data.

n_folds: the parameter that you can pass to tell the cross validation how many chunks to cut your data into (and then it will test each piece against the rest of the data).

```
from sklearn.grid_search import GridSearchCV

def cv_optimize(X, y, n_folds=10, num_p=100, penalty="l2"):
    clf = LogisticRegression(penalty)
    parameters = {"C": np.logspace(-4, 3, num=num_p)}
    gs = GridSearchCV(clf, param_grid=parameters, cv=n_folds)
    gs.fit(X, y)
    return gs
```

#This code block will do what we did earlier, which was fitting a line to the data we have, but it will do it many times with different sets of the data each time, to find the best possible fit out of many possible models.

(There are good wikipedia articles on all these things.)

Rahul recommends "Introduction to Information Retrieval" http://nlp.stanford.edu/IR-book/

A document is a collection of words. Some are useless, some are useful.
Useless words are often grouped into a list called Stop Words.

Corpus: a collection of documents

In a vector model, you're sort of thinking of the frequency of each word in the document as a value on a multidemensional axis (one dimension for each axis). You can imagine that documents with similar word distributions will be near each other on this multidimensional axis. It's easier to think of when you start with just two words in your distribution.

We can use the angles of these vectors to cluster documents according to their contents by measuring the angles between these vectors.

Vector space model is 20 or 30 years old, and is the basis for automating "if you liked x, you might like y" recommendations.

"tf–idf": term frequency–inverse document frequency

"Think Bayes" a very fun Oreilly book online at http://www.greenteapress.com/thinkbayes/

bayes theorem: http://en.wikipedia.org/wiki/Bayes'_theorem

regularization: adding information to prevent an non-useful outcome; ex.: alpha, in the naive bayes section of today's notebook

"What's the first thing you do with a new data set? You explore it."

There are parameters that you're setting in scikitlearn when doing distributions, but they have defaults. This can lead to overfitting even when you haven't intentionally set any variables.

#the grid of parameters to search over
alphas = [0, .1, 1, 5, 10, 50]
min_dfs = [1e-5, 1e-4, 1e-3, 1e-2, 1e-1] *# This sets increasingly restrictive word frequency filters, cutting out words making up larger and larger portions of the text*

#Find the best value for alpha and min_df, and the best classifier
best_alpha = None
best_min_df = None
maxscore=-np.inf
for alpha in alphas:
  for min_df in min_dfs:
    vectorizer = CountVectorizer(min_df = min_df)
    Xthis, ythis = make_xy(critics, vectorizer)

    #your code here
    clf = MultinomialNB(alpha=alpha)

```
cvscore = cv_score(clf, Xthis, ythis, log_likelihood)

if cvscore > maxscore:
    maxscore = cvscore
    best_alpha, best_min_df = alpha, min_df
```

read this if you have not yet: http://scott.fortmann-roe.com/docs/BiasVariance.html

look at "k-nearest neighbors" section of today's notebook on your own
interesting note: OpenRefine offers various nearest neighbor algorithms as clustering options

rmse: root mean square error - what's the rough error on each point

err on the side of choosing a model with more bias than variance - if your model doesn't predict anything you might as well go home.

more data = more complex models can be predictive

high variance situation = you could try getting more data
bias limited siatuated = "no amount of data is going to make a straight line fit better"

Suggested little project: Yelp Phoenix dataset chall
http://www.yelp.com/dataset_challenge/
write a naive-bayes classifier to try to predict whether a review is good or bad (decide where the cutoff point is, in number of stars)