There were some questions in class that I wasn't able to answer completely.

What are some examples of clustering?
How can the order of training data affect a classifier's predictions?
How does an SVM, an inherently two-class classifier, work to classify into multiple classes?
What is this business about predict requiring a 2D rather than 1D array?

Let me give those another shot here.

What are some examples of clustering?
An example inspired by a question late in class:
A university surveys incoming freshmen according to their interests/hobbies. It can assign orientation groups by using clustering to identify students with similar interests.

Another example of clustering from:
http://www.slideshare.net/EdurekaIN/applications-of-clustering-in-real-life
A cell carrier needs to decide where to locate towers to best serve its customers. It can use unsupervised learning to find clusters of collocated customers and build towers accordingly.

Clustering can be used any time you want to group similar observations/samples but you don't have pre-determined labels/classes. We don't know what sets of similar interests incoming freshmen will have, but we want them to be grouped by shared interests somehow. Cities aren't already organized into cell-tower-radius sized units, but somehow we need to assign each person to a cell-tower-radius sized area. Problems like these can be difficult to even define mathematically, and even when you can, they still can be difficult to solve. Clustering algorithms turn these problems into math in different ways and then attempt to solve them.

How can the order of training data affect a classifier's predictions?
A neural network is an example of a classifier that can be affected by the order of the training data. It is possible to train a neural network one observation at a time, adjusting the weights (internal parameters of a neural network) each time to better accommodate the new information. (This is similar, in a sense, to how the guess to a solution of equations is iteratively refined by Newton's method. The specifics may be different, but the idea that the values for the variables are iteratively changed to improve how well they solve the problem is the same.) The scheme for updating the weights is such that different orderings of the observations can result in different weights even if the same training data is used.

Linear SVMs work totally differently. They orient the hyperplanes optimally, considering all the information at once, which is why the order doesn't matter.

It just depends on the algorithm.

How does an SVM, an inherently two-class classifier, work for multiple classes?
You turn the multiple class problem into several two-class problems. Here's a good source, Wikipedia is another.

What I showed in class is one way (although my drawing wasn't correct):
for each class of data, generate a hyperplane that separates its observations from *all* other observations. This is called a "one-vs-all" approach.

Another approach:
for each possible *pair* of classes of data, generate a hyperplane that separates the pairs. This is called a "one-vs-one" approach.

To predict, you compare the new observation with *all* the hyperplanes, and use the results to decide which class it is in. (Details of this step are in the Wikipedia article, as it's possible for there to be conflicting information.)

Note that it's easier to find a "one-vs-one" hyperplane than a "one-vs-all" hyperplane, but the one-vs-one approach requires finding more hyperplanes (if there are more than three classes).

The drawing in class was fine for the first two classes, but wrong when I introduced the third class. There should have been three planes, the hyperplane between the first two classes would need to be replaced if it were one-vs-all, etc... Ask in discussion for new drawings if you're interested.

What is this business about predict *requiring a 2D rather than 1D array?*
Let's take a step back to numpy array slicing.

Consider a 2D (n_samples x n_features) array X.

I can select the ith row like:
`X[i,:]`
That returns the row as a 1D array of length n_features.

I can also select the ith row like:
`X[i]`
This also returns the row as a 1D array of length n_features. The ",:" which means "all columns" is implicit.

However, If I select the ith row like X[i:i+1,:] or X[i:i+1], this would return it as a 2D (1 x n_features) array.

Suppose I've trained a LinearSVC s using all but the last sample and want to test on *only* the last sample. You might think I could call predict like:
s.predict(X[-1])
but sklearn would give me an error:

```
DeprecationWarning: Passing 1d arrays as data is deprecated in 0.17 and will raise
ValueError in 0.19. Reshape your data either using X.reshape(-1, 1) if your data
has a single feature or X.reshape(1, -1) if it contains a single sample.
```

The point is that the data has to be 2D (1 x `n_features`), not 1D of length `n_features`. It also gives a *suggestion* for how to reshape your data to be a 2D array, but that is really up to you to decide how to do that.

So a solution is to write:
`s.predict(X[-1:])`
which is shorthand for:
`s.predict(X[len(X)-1:len(X)])`

Equivalently, I could use `sklearn`'s suggestion:

```
x = X[-1:]
s.predict(x.reshape((1,-1)))
```

However you do it, make sure you pass `predict` a 2D array.