0.1 LARS

Taking a more fundamental approach to regularization with LARS

To borrow from Gilbert Strang's evaluation of the Gaussian elimination, LARS is an idea you probably would've considered eventually had it not been discovered previously by Efron, Hastie, Johnstone, and Tibshiriani in their works[1].

0.1.1 Getting ready

Least-angle regression (LARS) is a regression technique that is well suited for high-dimensional problems, that is, $p >> n$, where p denotes the columns or features and n is the number of samples.

0.1.2 Implementation

First, import the necessary objects. The data we use will have 200 data points and 500 features. We'll also choose a low noise and a small number of informative features:

```
>>> from sklearn.datasets import make_regression
>>> reg_data, reg_target = make_regression(n_samples=200,
n_features=500, n_informative=10, noise=2)
```

Since we used 10 informative features, let's also specify that we want 10 nonzero coefficients in LARS. We will probably not know the exact number of informative features beforehand, but it's useful for learning purposes:

```
>>> from sklearn.linear_model import Lars
>>> lars = Lars(n_nonzero_coefs=10)
>>> lars.fit(reg_data, reg_target)
```

We can then verify that LARS returns the correct number of nonzero coefficients:

```
>>> np.sum(lars.coef_ != 0)
```

The question then is why it is more useful to use a smaller number of features. To illustrate this, let's hold out half of the data and train two LARS models, one with 12 nonzero coefficients and another with no predetermined amount. We use 12 here because

we might have an idea of the number of important features, but we might not be sure of the exact number:

```
>>> train_n = 100
>>> lars_12 = Lars(n_nonzero_coefs=12)
>>> lars_12.fit(reg_data[:train_n], reg_target[:train_n])
>>> lars_500 = Lars() # it's 500 by default
>>> lars_500.fit(reg_data[:train_n], reg_target[:train_n]);
```

Now, to see how well each feature fit the unknown data, do the following:

```
>>> np.mean(np.power(reg_target[train_n:] - lars_12.predict(reg_data
[train_n:]), 2))
31.527714163321001
>>> np.mean(np.power(reg_target[train_n:] - lars_500.predict(reg_data
[train_n:]), 2))
9.6198147535136237e+30
```

Look again if you missed it; the error on the test set was clearly very high. Herein lies the problem with high-dimensional datasets; given a large number of features, it's typically not too difficult to get a model of good fit on the train sample, but overfitting becomes a huge problem.

0.1.3 Theoretical Background

LARS works by iteratively choosing features that are correlated with the residuals. Geometrically, correlation is effectively the least angle between the feature and the residuals; this is how LARS gets its name.

After choosing the first feature, LARS will continue to move in the least angle direction, until a different feature has the same amount of correlation with the residuals. Then, LARS will begin to move in the combined direction of both features. To visualize this, consider the following graph: So, we move along x1 until we get to the point where the pull on x1 by y is the same as the pull on x2 by y. When this occurs, we move along the path that is equal to the angle between x1 and x2 divided by 2. There's more... Much in the same way we used cross-validation to tune ridge regression, we can do the same with LARS:

```
>>> from sklearn.linear_model import LarsCV
>>> lcv = LarsCV()
>>> lcv.fit(reg_data, reg_target)
```

Using cross-validation will help us determine the best number of nonzero coefficients to use. Here, it turns out to be as shown:

```
>>> np.sum(lcv.coef_ != 0)
44
```

[1]: Efron, Bradley; Hastie, Trevor; Johnstone, Iain and Tibshirani, Robert (2004). "Least Angle Regression". Annals of Statistics 32(2): pp. 407– 499. doi:10.1214/009053604000000067. MR 2060166.