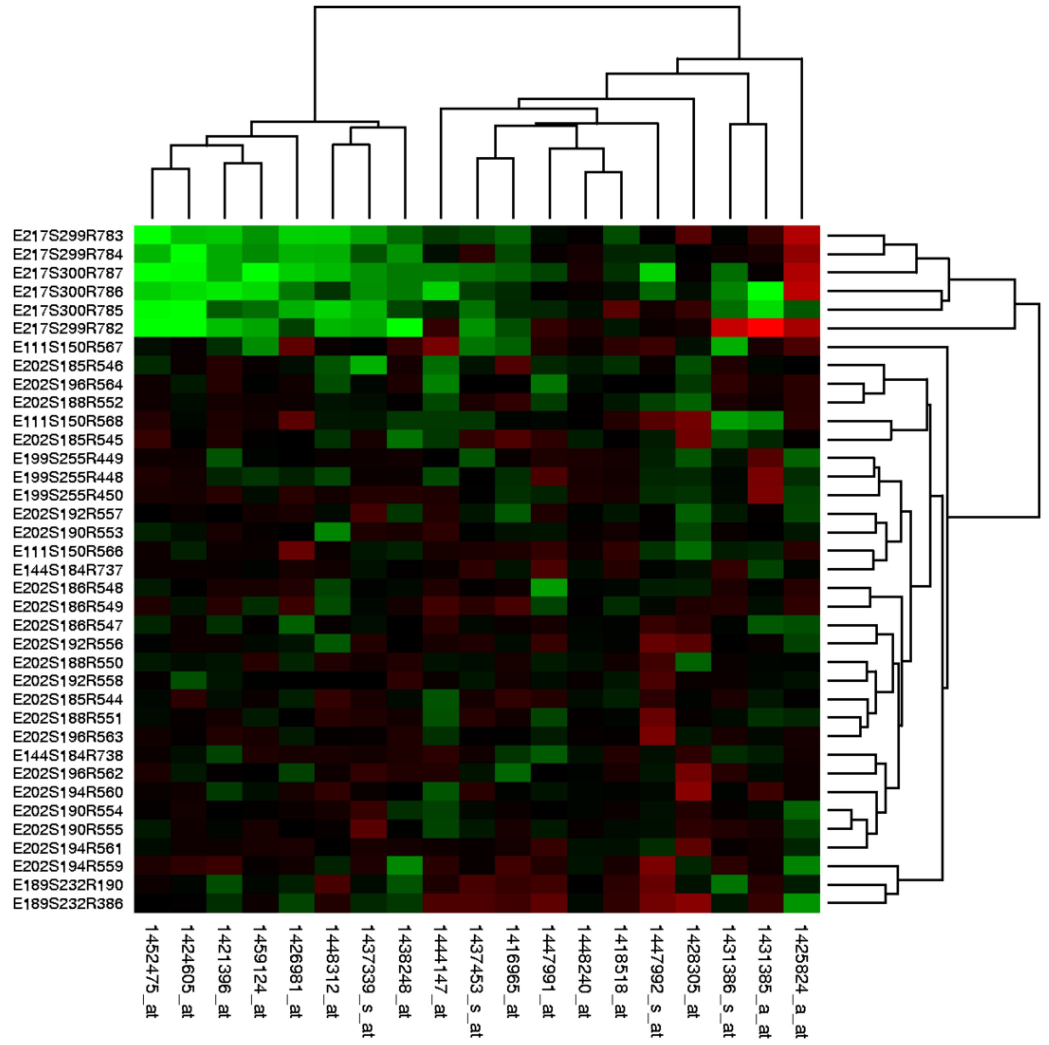
Sparse Linear Model in High Dimensions

Puyuan Yao

Mentor: Facheng Yu

High-dimensional Data

When setting up models for real-world scenarios, the dataset we study on are usually with high dimensions. For example, dataset for gene expression analysis and web search index. These high dimensional data allows us to form models that have good estimation for the real world. There are lots of examples of high dimensional data from the real world like gene expression.



Data Augmentation

To let low dimensional data has better performance compared to high dimensional data, we can do data augmentation to generate dimensions from original dimensions.

One common technique of data augmentation is through polynomial. By only 5th polynomial. We can generate over 1000 dimensions to the data.

Motivation of sparse linear model

Assume we want to estimate the parameter of a linear model with least squares estimator, and the data's dimension d is larger than the size of the data n, that is, d>>n.

Next, we want to calculate solutions by minimizing least square error and empirical least square error.

$$f(x) = x^T\theta, \theta \in \mathbb{R}^d$$

$$\theta^* = argmin_{\theta \in \mathbb{R}^d} E[(x^T\theta - y)^2]$$

$$\hat{\theta} = argmin_{\theta \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^{n} (x_i^T\theta - y_i)^2$$

Motivation of sparse linear model

By differentiating, we can get the solutions.

Since the data's dimension is larger than the size of the data, the first part of the solution to the ordinary least square estimator will not be invertible.

This can lead to either no solution, or infinitely many solutions.

$$\theta^* = argmin_{\theta \in \mathbb{R}^d} E[(x^T\theta - y)^2]$$

$$\theta^* = E[xx^T]^{-1} E[xy]$$

$$\hat{\theta} = argmin_{\theta \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^{n} (x_i^T\theta - y_i)^2$$

$$\hat{\theta} = (X^TX)^{-1}(X^TY)$$

Motivation of sparse linear model

$$S(\theta^*) := \{j \in \{1, \ldots, d\} : \theta^*_j \neq 0\}$$

To make ordinary least square estimator's solution having a unique solution, we can make an assumption about the parameter of the model that indicates only some of the parameters are non-zero, which is the sparsity assumption.

The hard sparsity requires L1 norm of S is substantially smaller than d. Under the sparsity assumption, we may have a unique linear solution of the least square.

Lasso program

Consider a linear model with a noise vector w.

By sparsity assumption, we want to minimize the square error while controlling the sparsity of the parameters.

Using Lagrangian method, we can write above two constraints in one minimization problem.

$$y = \mathbf{X}\theta^* + w$$

$$\min_{\theta \in \mathbb{R}^d} \left\{ \frac{1}{2n} ||y - \mathbf{X}\theta||_2^2 + \lambda_n ||\theta||_1 \right\}$$

Consistency of Lasso program

RE condition

$$\frac{1}{n}\|\mathbf{X}\Delta\|_2^2 \geq k\|\Delta\|_2^2 \text{ for all } \Delta \in \mathbb{C}_3(S)$$

$$\|\hat{\theta} - \theta^*\|_2 \leq \frac{3}{k}\sqrt{s}\lambda_n$$

with

$$\lambda_n \geq 2\|\frac{\mathbf{X}^T w}{n}\|_\infty$$

In addition to hard sparsity assumption, we also need to introduce RE condition. The RE condition provides a guarantee that these algorithms can distinguish the truly important predictors from the irrelevant ones, even when the data involves many variables that are correlated with each other.

By the two assumptions, we are guaranteed an upper bound of the difference between estimation and true parameter.

Application

Since the lasso program constrains the number of non-zero parameters, we can use it for feature selection.

We suppose that the selected feature behaves similarly as the original features.

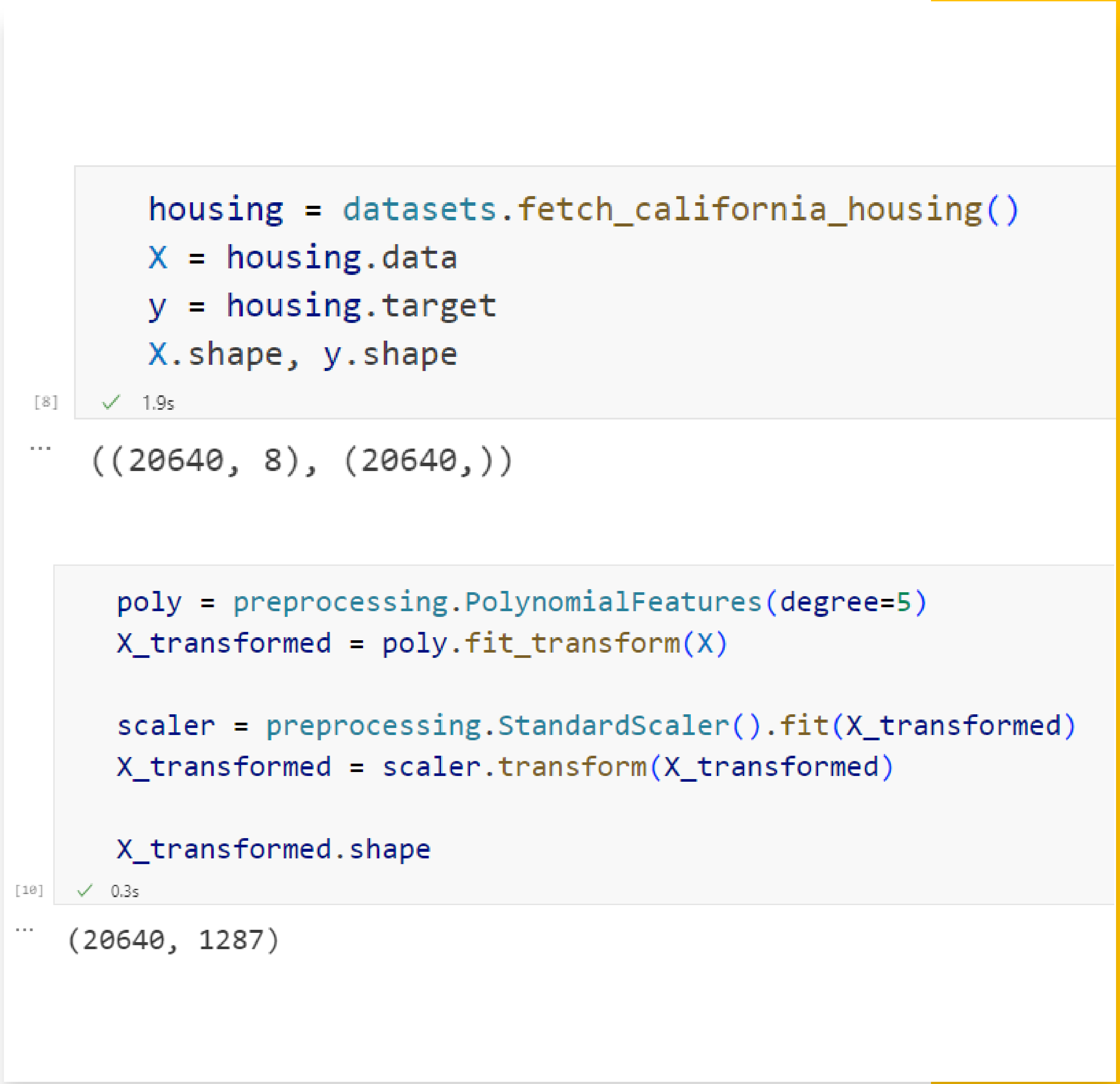
$$E[(Y_i - X_i^\top \theta_s^*)^2] = E[(Y_i - X_i^\top \theta^*)^2]$$

Real world example

One of the common application of linear model is predicting house price.

In this example, we will use the real-world data of California house price to demonstrate how lasso program can help make prediction more accurate.

Since we want to use lasso program to select most efficient feature, we can apply data augmentation to generate lots of features, and later select features among them.



```
housing = datasets.fetch_california_housing()
X = housing.data
y = housing.target
X.shape, y.shape
```

[8] ✓ 1.9s

```
((20640, 8), (20640,))
```

```
poly = preprocessing.PolynomialFeatures(degree=5)
X_transformed = poly.fit_transform(X)

scaler = preprocessing.StandardScaler().fit(X_transformed)
X_transformed = scaler.transform(X_transformed)

X_transformed.shape
```

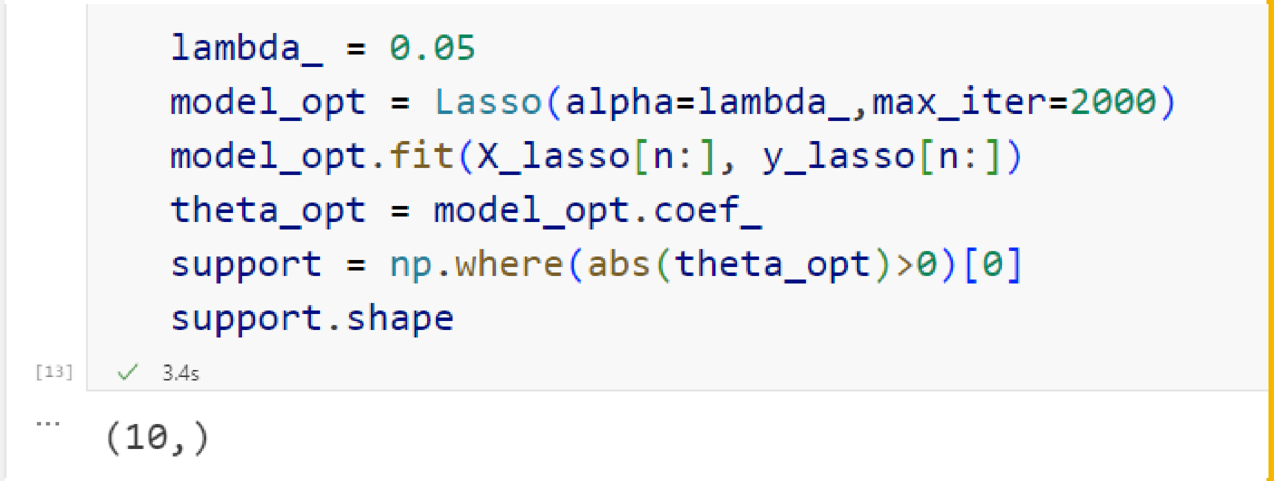
[10] ✓ 0.3s

```
(20640, 1287)
```

Real world example

By running lasso program, there are 10 features that were selected.

We can compare the mean squared error between the models with raw features and selected features.



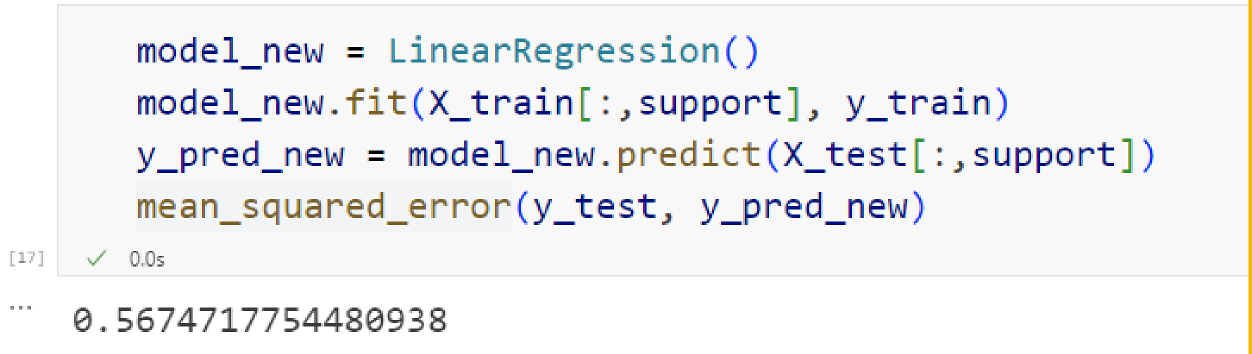
```
lambda_ = 0.05
model_opt = Lasso(alpha=lambda_,max_iter=2000)
model_opt.fit(X_lasso[n:], y_lasso[n:])
theta_opt = model_opt.coef_
support = np.where(abs(theta_opt)>0)[0]
support.shape
```

```
(10,)
```

Real world example

The model with selected features has less mean squared error comparing to the model with original features.

Selected features

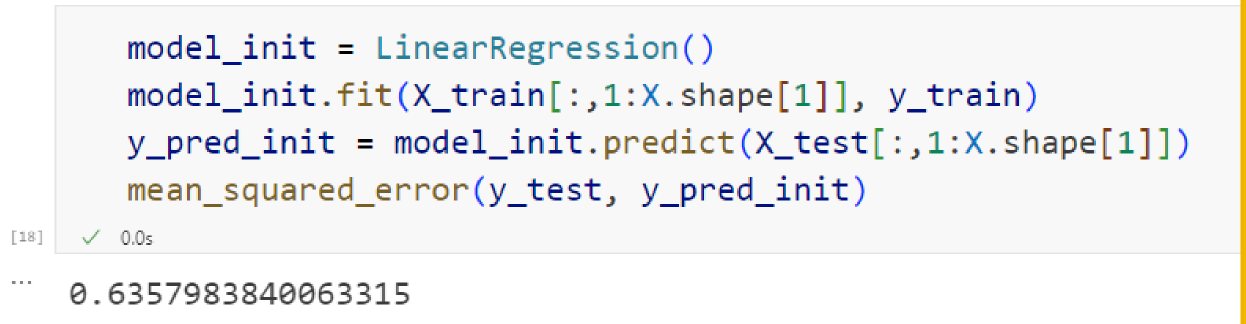


```
model_new = LinearRegression()
model_new.fit(X_train[:,support], y_train)
y_pred_new = model_new.predict(X_test[:,support])
mean_squared_error(y_test, y_pred_new)
```

[17] ✓ 0.0s

```
0.5674717754480938
```

Original features



```
model_init = LinearRegression()
model_init.fit(X_train[:,1:X.shape[1]], y_train)
y_pred_init = model_init.predict(X_test[:,1:X.shape[1]])
mean_squared_error(y_test, y_pred_init)
```

[18] ✓ 0.0s

```
0.6357983840063315
```

Thank you